robotron

ANWENDERDOKUMENTATION

FORTRAN
UDOS-1526

2. Auflage

Fortran
für Bürocomputer A 5120/30
mit dem Betriebssystem
UDOS 1526

VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt 1984

Die vorliegende Dokumentation entspricht dem Stand vom 1.2.1984.
Zum System "UDOS 1526" gehören per 31.1.84 insgesamt folgende Dokumentationen:

- Systemhandbuch UDOS 1526
- Sprachbeschreibungen BASIC
  PASCAL
  FORTRAN
  Z8ASM
  PLZ/SYS
  SYSTEM U 8000

- Beschreibung zum REASSEMBLER (REASM)
- Beschreibung zum Symbolischen Debugger (SYD)

Nachdruck, jegliche Vervielfältigungen dieser Unterlage oder Auszüge daraus sind unzulässig.

Die Ausarbeitung dieser Dokumentation erfolgte durch ein Kollektiv des VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt.

Herausgeber: VEB Robotron-Buchungsmaschinenwerk
Karl-Marx-Stadt
9010 Karl-Marx-Stadt
PSF 129

Die Entwickler/Herausgeber sind dankbar für Kritiken, Hinweise und Wünsche, die bitte schriftlich an den Bereich "software-Zentrum" zu übergeben sind.

Inhaltsverzeichnis

		Seite
1.	Einführung	2
2.	Form und Sprachelemente von FORTRAN-Programmen	2
2.1.	FORTRAN-Zeichenmenge	2
2.2.	Zeilenaufbau	3
2.3.	Zeilentypen	3
2.4.	Marken (Labels)	3
2.5.	Anweisungen	4
2.6.	Datentypen	4
2.7.	Konstante	5
2.8.	Variable	5
2.9.	Felder und Feldelemente, Indizes	6
3.	Ausdrücke	6
3.1.	Arithmetische Ausdrücke	6
3.2.	Logische Ausdrücke	7
4.	Anweisungen	8
4.1.	Ergibt-Anweisungen	8
4.2.	Steueranweisungen	8
4.2.1	GOTO-Anweisungen	9
4.2.2	IF-Anweisungen	9
4.2.3	DO- oder Schleifen-Anweisung	10
4.2.4	STOP- oder PAUSE-Anweisung	11
4.3.	Vereinbarungsanweisungen	11
4.3.1	Typvereinbarungen	12
4.3.2	DIMENSION-(oder Feld-)Anweisungen	12
4.3.3	COMMON-Anweisungen	13
4.3.4	EQUIVALENCE-Anweisungen	14
4.3.5	DATA-Anweisung	15
4.3.6	EXTERNAL-Anweisung	15
5.	Ein- und Ausgabeanweisungen	16
5.1.	Formatgebundene READ- und WRITE-Anweisungen	16
5.2.	Formatfreie READ- und WRITE-Anweisungen	18
5.3.	ENCODE- und DECODE-Anweisungen	18
5.4.	FORMAT-Anweisungen	18
5.4.1	I-Format	19
5.4.2	E-, D-, F-Format	20
5.4.3	L-Format	21
5.4.4	G-Format	21
5.4.5	Wirkung des Skalenfaktors uP	22
5.4.6	A-Format	22
5.4.7	Literalformate	23
5.4.8	X-Format	23
5.5.	Spezielle Steuereigenschaften von Format-Anweisungen	24
5.6.	Diskfile-Zugriff	25
5.7.	I/O-Interface	26
6.	Unterprogramme und Funktionen	27
6.1.	Anweisungsfunktionen	28
6.2.	FUNCTION-Unterprogramme	28
6.3.	SUBROUTINE-Unterprogramme	30
6.4.	Verarbeitung von Feldern in Unterprogrammen	31
6.5.	Standard-Unterprogramme	31
6.6.	Einbinden von Unterprogrammen	31
6.7.	BLOCK DATA-Unterprogramme	32

7.	Aufbau von U880-FORTRAN-Programmen	33
8.	Compilierung von FORTRAN-Programmen	34
8.1.	Kommandoformat	34
8.2.	Fehlermeldungen des FORTRAN-Compilers	36
9.	Binden von FORTRAN-Objektprogrammen	37
9.1.	Der Binder (FORTRAN-LINKER) FLINK	37
9.2.	FLINK-Fehlermeldungen	39
9.3.	Laufzeit-Fehlermeldungen	40
10.	Standardfunktionen	42

1. Einführung

FORTRAN ist eine universelle problemorientierte Programmiersprache. Der Name ist ein Akronym für FORmula TRANslator. Die syntaktischen Regeln der Sprache sind streng und fordern von Programmierer, daß er seine Probleme in einer Reihe exakter Anweisungen formuliert. Diese Anweisungen, genannt das Quellprogramm, werden durch einen Compiler in die Maschinensprache des Computers übersetzt, auf dem das Programm ausgeführt werden soll.
Der vorliegende Text beschreibt die FORTRAN-Quellsprache für Mikrorechner auf der Basis des Mikroprozessors U880. Gegenüber dem Standard-FORTRAN gibt es einige Erweiterungen und Einschränkungen, die aus dem Text hervorgehen. Der COMPLEX-Datentyp ist im U880-FORTRAN nicht enthalten.

2. Form und Sprachelemente von FORTRAN-Programmen

FORTRAN-Quellprogramme bestehen aus Programmeinheiten: aus einem Hauptprogramm (MAIN-Programm) und einer Anzahl von Unterprogrammen. Eine Programmeinheit enthält nichtausführbare (Vereinbarungs-) und ausführbare Anweisungen. Jede Anweisung besteht aus Zeichen der zugelassenen FORTRAN-Zeichenmenge und hat einen vorgeschriebenen Zeilenaufbau.

2.1. FORTRAN-Zeichenmenge

Die Zeichenmenge besteht aus alphanumerischen Zeichen, den

Buchstaben: A,B,C,D,...X,Y,Z und den
Ziffern : Ø,1,2,3,...8,9

Bei Buchstaben wird nicht zwischen Groß- und Kleinbuchstaben unterschieden. Zugelassen sind weiter die

Hexadezimalziffern: 'Ø,1,2,...8,9,A,B,C,D,E,F'H

Es sind die folgenden Sonderzeichen zugelassen :

b	blank (space)	/	Schrägstrich
=	Gleichheitszeichen	(	öffnende Klammer
+	Pluszeichen	)	schließende Klammer
-	Minuszeichen	,	Komma
*	Stern	.	Dezimalpunkt
			Apostroph

2.2. Zeilenaufbau

Eine Zeile besteht aus max. 80 Zeichen (Spalten), die in vier Felder zusammengefaßt sind.

1.	Spalten 1-5	Markenfeld (Label)
2.	Spalte 6	Fortsetzungszeichenfeld
3.	Spalten 7-72	Anweisungsfeld
4.	Spalten 73-80	Identifikationsfeld

Das Identifikationsfeld kann beliebig genutzt werden. Der Inhalt dieser Spalten wird vom Compiler ignoriert.

2.3. Zeilentypen

Es gibt vier verschiedene Zeilentypen.

1. Kommentarzeile: Die Spalte 1 enthält den Buchstaben C. Die Spalten 2-72 können für Kommentare genutzt werden oder leer bleiben. Kommentarzeilen haben keinen Einfluß auf das Quellprogramm.

2. END-Zeile: Die Spalten 1 bis 5 können eine Marke enthalten. Spalte 6 muß 0 oder b enthalten. Die Spalten 7-72 enthalten die Zeichen E,N,D (in dieser Reihenfolge). Jede FORTRAN-Programmeinheit muß als letzte Zeile eine END-Zeile haben, um dem Compiler das physische Ende der Programmeinheit mitzuteilen.

3. Anfangszeile: Die einzige oder erste Zeile einer Anweisung. Die Spalten 1-5 können eine Marke enthalten, um die Zeile zu kennzeichnen. Spalte 6 muß 0 oder b enthalten. Die Spalten 7-72 enthalten die gesamte oder Teile einer Anweisung. Die Anweisung muß nicht in Spalte 7 beginnen.

4. Fortsetzungszeile: Sie wird verwendet, wenn sich eine Anweisung über mehr als eine Zeile erstreckt. Die Spalten 1-5 werden ignoriert. Wenn Spalte 1 ein C enthält ist es eine Kommentarzeile. Spalte 6 muß ein von 0 und b verschiedenes Zeichen enthalten. Die Spalten 7-72 enthalten die Fortsetzung der Anweisung. Die Anzahl der Fortsetzungszeilen ist nicht beschränkt.

2.4. Marken (Labels)

In den Spalten 1 bis 5 einer Anfangszeile kann eine Marke enthalten sein. Auf die Marke kann in anderen Anweisungen Bezug genommen werden.

Eine Marke ist eine natürliche Zahl von 1 bis 99999.
Der numerische Wert einer Marke, führende Nullen oder ß sind irrelevant.
Eine bestimmte Marke darf nur einmal im Markenfeld auftreten. Eine Marke in einer Fortsetzungszeile wird vom Compiler ignoriert.

2.5. Anweisungen

Die Anweisungen werden eingeteilt in ausführbare und nichtausführbare Anweisungen.
Ausführbare Anweisungen spezifizieren Aktionen und werden vom Compiler in Objektprogrammbefehle umgesetzt. Es gibt drei Arten ausführbarer Anweisungen:

Ergibtanweisungen,
Steueranweisungen,
Ein-/Ausgabeanweisungen.

Nichtausführbare Anweisungen beschreiben dem Compiler die Art und Anordnung von Daten und geben Informationen über Ein- und Ausgabeformate und Dateninitialisierungen an das Objektprogramm während des Ladens und der Ausführung von Programmen.
Es gibt fünf Arten nichtausführbarer Anweisungen:

Vereinbarungsanweisungen,
Dateninitialisierungsanweisungen,
Formatanweisungen,
Anweisungsfunktionsdefinitionen,
Unterprogrammanweisungen.

2.6. Datentypen

Die von FORTRAN-Programmen verarbeiteten Daten (Konstanten und Variable) können zu einem der folgenden Datentypen gehören:

INTEGER	ganze Zahl aus dem Bereich -32768, 32767 ;
REAL	reelle Zahl, die im Gleitkommaformat 4 Bytes umfaßt. Allgemeine Form ± i.fE±e, wobei i, f und e vorzeichenlose natürliche Zahlen sind. Für e gilt $-38 \leq e \leq +38$;
DOUBLE PRECISION	reelle Zahl, die im Gleitkommaformat 8 Bytes umfaßt. Allgemeine Form ±i.fD±e, mit i,f und e wie bei REAL ;
LOGICAL	Typ der logischen (Booleschen) Werte TRUE und FALSE, wobei .FALSE.=Ø und .TRUE.≠Ø ist.

Es gelten die folgenden Äquivalenzen bei Typbezeichnungen:

BYTE, INTEGER*1, LOGICAL*1, LOGICAL	sind äquivalent
INTEGER*2, LOGICAL*2, INTEGER	sind äquivalent
REAL, REAL*4	sind äquivalent
DOUBLE PRECISION, REAL*8	sind äquivalent.

Der für das Abspeichern einer REAL-Größe erforderliche Speicherplatz von 4 Bytes heißt Speichereinheit. Die einzelnen Typen belegen die folgenden Anzahlen von Speichereinheiten:

INTEGER	1/2	Speichereinheit = 2 Bytes (Typenlänge 2),
LOGICAL	1/4	Speichereinheit = 1 Bytes (Typenlänge 1),
REAL	1	Speichereinheit = 4 Bytes (Typenlänge 4),
DOUBLE PRECISION	2	Speichereinheit = 8 Bytes (Typenlänge 8).

Bei der 4 Bytes-Darstellung der REAL-Größen enthält das erste Byte den Exponenten. Der Wert 8ØH entspricht dem Exponenten Ø. Werte < 8ØH entsprechen negativem, Werte > 8ØH entsprechen positivem Exponenten. Ist der Exponent Ø, dann ist die ganze Zahl Ø.

2.7. Konstante

Konstante können einen der in 2.6. behandelten Typen haben. Sie können weiter Zeichenkettenkonstante sein. Zeichenketten haben folgende Formen:

a) In Apostroph eingeschlossene Zeichenfolgen, z.B. 'TEXT'. Soll in der Zeichenfolge ein Apostroph vorkommen, muß er doppelt geschrieben werden, z.B. 'GEHT''S' entspricht der Zeichenfolge GEHT'S.

b) Durch Angabe der Anzahl der Zeichen der Zeichenfolge, den Buchstaben H und die Zeichenfolge, z.B. 4HTEXT, 6HGEHT'S. In dieser Darstellung zählen Leerzeichen mit, z.B. 8H␣L␣E␣E␣R für L E E R .

Konstante können auch hexadezimale Zahlen ohne Vorzeichen sein. Sie werden in der Form

Z'hhhh' oder X'hhhh'

dargestellt, wobei h eine Hexadezimalziffer ist. Z.B. X'A13F'.

2.8. Variable

Variable werden durch symbolische Namen gekennzeichnet. Die Namen bestehen aus 1 bis 6 alphanumerischen Zeichen, wobei das erste Zeichen ein Buchstabe sein muß. Unter dem Wert einer Variablen versteht man den entsprechend des vereinbarten Typs interpretierten aktuellen Inhalt der zugehörigen Speicherstelle. Variable haben einen Typ und eine Typlänge (Abschnitt 2.6.). Die Festlegung des Typs erfolgt in einer der folgenden Arten:

1. Implizite Festlegung: Wenn nicht anders festgelegt, stellen symbolische Namen, die mit I, J, K, L, M, N beginnen INTEGER-Variable dar. Alle anderen Namen bezeichnen REAL-Variable.

2. Explizite Festlegung: Variablentypen können explizit durch Typvereinbarungen festgelegt werden (Abschnitt 4.3.1).

Variablen werden während der Programmabarbeitung numerische Werte zugewiesen. Durch eine DATA-Anweisung kann ebenfalls ein Wert zugewiesen werden (Abschnitt 4.3.5). Zeichenketten können jedem Variabletyp zugewiesen werden (Abschnitt 5.4.6).

2.9. Felder und Felderelemente, Indizes

Ein Feld ist eine geordnete Menge von Daten der Dimensionen 1, 2 oder 3. Es wird durch einen symbolischen Namen in der gleichen Weise wie eine Variable beschrieben, ausgenommen daß ein Feldnahme durch eine Feldvereinbarung festgelegt werden muß. Ein Feldelement ist ein Element der Datenmenge, die das Feld bildet. Auf ein Feldelement wird durch die Indizierung des Feldnamens Bezug genommen. Der Begriff "Feldelement" ist gleichbedeutend mit dem Begriff "indizierte Variable" (Abschnitt 4.3.2). Durch eine DATA-Anweisung kann jedem Feldelement ein Anfangswert zugewiesen werden. Während der Programmabarbeitung können Feldelementen numerische Werte zugewiesen werden.
Der Index eines Feldelementes besteht aus 1, 2 oder 3 in Klammern eingeschlossenen Indexausdrücken, die jeweils durch Komma getrennt sind. Die Anzahl der Indexausdrücke muß mit der in der Feldvereinbarung festgelegten Dimension übereinstimmen, ausgenommen in EQUIVALENCE-Anweisungen (Abschnitt 4.3.4).
Bezeichnen M und N Integerkonstante und L eine Integervariable, dann sind folgende Indexausdrücke zugelassen:

	z.B.	
M*L+N		3*I+5
M*L-N		7*K-3
M*L		3*J
L+N		J+6
L-N		J-27
L		J
N		13

Indizes selbst können nicht indiziert werden.
Beispiele von Feldelementen:

X(2*J-3,7), A(I,J,K), I(2Ø), C(L-2), Y(I).

3. Ausdrücke

Die Ausdrücke in FORTRAN sind das Analogon zu den Formeln in der Mathematik. Mit ihnen lassen sich Werte berechnen. Ein Ausdruck besteht aus einem einzigen Operanden oder aus mehreren durch Operatoren verknüpften Operanden. Es gibt arithmetische und logische Ausdrücke.

3.1. Arithmetische Ausdrücke

Als Operatoren sind zugelassen:

\+ (Addition), - (Subtraktion), * (Multiplikation), / (Division) und ** (Potenzierung).

Als Operanden sind zugelassen:

Konstante, Variable, Feldelemente, Funktionen.

In Ausdrücken sind Klammern zugelassen (Anzahl der öffnenden gleich Anzahl der schließenden Klammern).
Der Wert eines Ausdrucks kann vom Typ INTEGER, REAL, DOUBLE PRECISION oder LOGICAL sein. Der Typ wird durch die Datentypen der Operanden des Ausdrucks bestimmt. Sind die Elemente nicht alle vom gleichen Typ, dann wird der Typ des

Ausdrucks durch das Element mit dem höchsten Typ bestimmt.
Die Typhierarchie ist (vom höchsten zum niedrigsten):

DOUBLE PRECISION, REAL, INTEGER, LOGICAL.

Arithmetische Ausdrücke werden in der folgenden Reihenfolge berechnet:

1. Klammern (bei Schachtelung zuerst die innerste Klammer),
2. Funktionsaufruf,
3. Potenzieren
4. Multiplikation und Division,
5. Addition und Subtraktion.

Bei Operation auf gleicher Ebene verläuft die Berechnung von links nach rechts.

A**(Z-((X+Y)/T))**J

wird berechnet: X+Y=B1, B1/T=B2, Z-B2=B3, B3**J=B4,
A*B4=B5=Wert des Ausdrucks.

Der Ausdruck X**Y**Z ist nicht erlaubt. Entweder

(X**Y)**Z oder X**(Y**Z).

Indexausdrücke werden ebenfalls nach diesen Regeln berechnet.

3.2. Logische Ausdrücke

Der Wert eines logischen Ausdrücks ist .TRUE. oder .FALSE.
Ein logischer Ausdruck kann sein:

1. eine Verknüpfung logischer Operanden mit logischen Operatoren;
2. zwei durch einen Vergleichsoperator verknüpfte arithmetische Ausdrücke (Vergleichsausdruck).

Logische Operanden sind logische Konstante, logische Variable, logische Feldelemente, logische Funktionen, Vergleichsausdrücke.

Logische Operatoren sind: .NOT., .AND., .OR., .XOR.

U, V seien logische Ausdrücke. Damit ist

U	V	.NOT.U	U.AND.V	U.OR.V	U.XOR.V
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Zwei logische Operatoren dürfen nicht direkt nebeneinander stehen, ausgenommen der zweite Operator ist .NOT.

A .AND. .NOT.B ist erlaubt,
A .AND. .OR.B ist nicht erlaubt.

Es gibt sechs Vergleichsoperatoren

.LT.	kleiner als	$<$
.LE.	kleiner oder gleich	$\leq$
.EQ.	gleich	$=$
.NE.	nicht gleich	$\neq$
.GT.	größer als	$>$
.GE.	größer oder gleich	$\geq$

Beispiele: A .EQ. B; (A**J).GT.(ZAP*(RHO*TAU-ALEPH));
.NOT.-3.LT.I.AND..NOT.+8.LT.J; A.LT.B.AND.B.LT.C .

Logische Ausdrücke werden in der folgenden Reihenfolge berechnet:

1. Klammern (bei Schachtelung , zuerst die innerste Klammer);
2. Funktionsaufrufe,
3. Potenzieren
4. Multiplikation und Division;
5. Addition und Subtraktion;
6. .LT., .LE., .EQ., .NE., .GT., .GE. ;
7. .NOT. ,
8. .AND. ,
9. .OR., .XOR. .

In Ausdrücken können Zeichenketten- und Hexadezimal-Konstante anstelle von Integerkonstanten auftreten. Sie können deshalb nur 2 Byte lang sein. Ausnahmen sind:

1. Lange Zeichenkettenkonstante können als Unterprogrammparameter verwendet werden.
2. Zeichenketten- und Hexadezimalkonstante können in DATA-Anweisungen bis zu
   4 Byte lang sein, wenn sie REAL-Variablen und
   8 Byte lang sein, wenn sie DOUBLE PRECISION-Variablen zugeordnet sind.

4. Anweisungen

4.1. Ergibt-Anweisungen

Die allgemeine Form einer Ergibt-Anweisung ist

V = A

wobei V eine Variable oder ein Feldelement ist. Bei der Ausführung wird der Wert des Ausdrucks A berechnet und der Variablen oder dem Feldelement V zugeordnet. Eine Ergibt-Anweisung ist also keine Gleichung im mathematischen Sinne. Der Typ von V und A müssen nicht übereinstimmen. Vor der Zuordnung des Wertes von A zu V wird bei unterschiedlichen Typen eine Typwandlung durchgeführt. Dabei können Teile der Information verlorengehen.
Bei Ergibtanweisungen muß V= auf der gleichen Zeile stehen. Das gilt auch, wenn die Anweisung Teil einer logischen IF-Anweisung ist (Abschnitt 4.2.2).

Beispiel:
```
      A(5,3)=
1     BC(7,2)+SIN(C)
```

4.2. Steueranweisungen

Im allgemeinen werden in einem FORTRAN-Programm ausführbare Ausführungen zur Laufzeit in der gleichen Reihenfolge abgearbeitet, in der sie physisch im Quellprogramm auftreten. Mit Steueranweisungen kann dieser Programmablauf geändert werden.

4.2.1. GOTO-Anweisungen

Es gibt 3 verschiedene GOTO-Anweisungen:

1. Unbedingtes GOTO. Allgemeine Form: GOTO k
wobei k eine Anweisungsmarke einer ausführbaren Anweisung in derselben Programmeinheit ist. Das Programm wird mit der Anweisung mit der Marke k fortgesetzt.

2. Berechnetes GOTO. Allgemeine Form: GOTO (k1,k2,..,kn),j
wobei die ki Anweisungsmarken sind und j eine Integervariable im Bereich $1 \leq j \leq n$ ist. Das Programm wird mit der Anweisung mit der Marke kj fortgesetzt. Ist $j < 1$ oder $j > n$, wird mit der Anweisung nach GOTO fortgesetzt.

3. Gesetztes GOTO. Allgemeine Form: GOTO j,(k1,k2,..,kn)
wobei j der Name einer Integervariablen (Marke) ist und die ki Marken ausführbarer Anweisungen sind. Die Reihenfolge der ki ist unwichtig. Dieser Sprunganweisung muß eine ASSIGN-Anweisung vorausgehen. Sie hat die Form

```
ASSIGN J TO I
```

wobei J eine Marke und I eine Integervariable ist. Die Marke J wird der Variablen I zugewiesen.

Beispiel:
```
ASSIGN 17 TO LABEL
.
.
GOTO   LABEL,(34,37,1,42,17)
.
.
```

Das Programm wird an der Marke 17 fortgesetzt.

Äquivalent dazu wäre

```
LABEL = 5
.
.
GOTO  (34,37,1,42,17),LABEL
```

4.2.2. IF-Anweisungen

Arithmetische IF-Anweisung. Allgemeine Form: IF (A) m1,m2,m3
wobei A ein arithmetischer Ausdruck und die mi Marken ausführbarer Anweisungen der gleichen Programmeinheit sind. Die der arithmetischen IF-Anweisung folgende Anweisung muß eine Marke haben. Je nach Wert des Ausdruckes A wird das Programm bei m1,m2 oder m3 fortgesetzt.

```
A< Ø  Fortsetzung bei Marke m1
A= Ø  Fortsetzung bei Marke m2
A> Ø  Fortsetzung bei Marke m3
```

Beispiel: IF (D+(A/B)**3) 7,1Ø,21

Logische IF-Anweisung. Allgemeine Form: IF (u),s
wobei u ein logischer Ausdruck und s eine ausführbare Anweisung ist. Eine DO-Anweisung (Abschnitt 4.2.3) oder eine andere logische IF-Anweisung ist nicht zugelassen. Hat u den Wert

TRUE,wird s ausgeführt und mit der Anweisung nach s fortgesetzt, wenn s keine Sprunganweisung ist. Hat n den Wert FALSE, wird mit der Anweisung nach s fortgesetzt. Ist s eine Ergibt-Anweisung (V = A), dann muß entweder "V=" auf der selben Zeile wie IF(n)stehen, oder auf einer Fortsetzungszeile, wobei dann hinter IF(n) nur Leerzeichen stehen dürfen.

Beispiel:
```
      IF (I.GT.2Ø) GOTO 115
      IF (Z)   Z=A+B*(X-Y)     oder
      IF (Z)
    1 Z=A+B*(X-Y)
```

4.2.3. DO- oder Schleifen-Anweisung

Die DO-Anweisung ermöglicht das wiederholte Abarbeiten einer Folge von Anweisungen, des sogenannten DO-Bereiches.
Allgemeine Form: DO k i=m1,m2,m3
wobei k die Marke einer ausführbaren Anweisung, i eine ganzzahlige oder logische Variable und m1,m2 und m3 ganzzahlige Konstante oder Variable sind. Falls m3=1 ist, kann m3 weggelassen werden. i ist die Laufvariable, m1 der Anfangswert, m2 der Endwert und m3 die Schrittweite. Die mit k markierte Anweisung ist die letzte Anweisung des DO-Bereiches. Diese letzte Anweisung darf kein arithmetisches IF, GOTO, RETURN, STOP, PAUSE oder ein anderes DO sein. Ist die letzte Anweisung ein logisches IF mit dem Wert .FALSE., dann werden die Anweisungen des DO-Bereiches wiederholt. Bei .TRUE. wird die Anweisung des logischen IF ausgeführt und dann die Anweisungen des DO-Bereiches wiederholt. Die Anweisung des logischen IF darf kein GOTO, arithmetisches IF, RETURN, STOP oder PAUSE sein. Die Laufvariable muß positiv sein und darf durch Anweisungen des DO-Bereiches nicht verändert werden. Ebenso dürfen m1, m2 und m3 nicht verändert werden.
Sind m1, m2 und m3 vom Typ INTEGER*1,dann wird das Objektprogramm für die Schleife kürzer sie wird schneller abgearbeitet. Es sind dann allerdings nur max. 127 Operationen möglich. Die Anweisungen des DO-Bereiches werden immer wenigstens einmal abgearbeitet, auch wenn m1 < m2 ist. Ist i > m2, dann wird mit der Anweisung des DO-Bereiches fortgesetzt.
DO und die Angaben bis zum ersten Komma (einschließlich) müssen auf der Anfangszeile stehen.
Wenn die eigentliche Anweisung eines DO-Bereiches eine nicht zugelassene Form hat, wird als letzte Anweisung CONTINUE verwendet. CONTINUE ist eine Leeranweisung.

Beispiel:
```
      DO 27 I=1,25
      A(I)=B(I)**I
      IF(I.EQ.12)GOTO 115
   27 CONTINUE
      :
  115 N=I
      :
```

Eine logische IF-Anweisung mit einer Sprunganweisung ist als letzte Anweisung eines DO-Bereiches nicht zugelassen.
Die Anweisungen eines DO-Bereiches, die logisch zwischen DO und der letzten Anweisung ausgeführt werden, müssen nicht alle physisch zwischen DO und letzter Anweisung liegen. Teile des logischen DO-Bereiches können also außerhalb des physischen DO-Bereiches liegen. Man spricht dann von einem erweiterten Bereich.
Aus einem DO-Bereich kann herausgesprungen werden. In einen DO-Bereich darf nicht eingesprungen werden (Ausnahme: Rücksprung aus dem erweiterten Bereich).
DO-Anweisungen können geschachtelt werden. Die innere DO-Anweisung muß vollständig in der äußeren enthalten sein. Mehrere DO-Anweisungen können die gleiche letzte Anweisung haben. Zu dieser letzten Anweisung darf nur von der innersten DO-Schleife gesprungen werden.

4.2.4. STOP- und PAUSE-Anweisung

Die STOP-Anweisung ist das logische Ende eines Programms.

Allgemeine Form: STOP C
wobei C eine Zeichenkette von maximal 6 Zeichen ist. C kann auch fehlen. Wird bei der Programmabarbeitung STOP C erreicht, erscheint C auf dem Bildschirm und das Programm wird beendet.
Durch die PAUSE-Anweisung kann die Ausführung eines Programms unterbrochen werden.

Allgemeine Form: PAUSE C (C wie oben)
Wird bei der Programmabarbeitung PAUSE C erreicht, wird C auf dem Bildschirm angegeben und das Programm gestoppt. Durch Eintasten von "T" wird die Programmabarbeitung beendet. Das Drücken einer anderen Taste startet die Programmausführung mit der hinter PAUSE stehenden Anweisung.

4.3. Vereinbarungsanweisungen

Vereinbarungsanweisungen sind nichtausführbare Anweisungen, die Datentypen von Variablen, Feldern und Felddimensionen definieren, die Speicherplatzzuordnungen festlegen und die variablen Daten Anfangswerte zuordnen. Es gibt sechs Arten von Vereinbarungsanweisungen.

Typ-, EXTERNAL- und DIMENSION- Anweisungen,
COMMON - Anweisungen,
EQUIVALENCE - Anweisungen,
DATA - Anweisungen.

Alle Vereinbarungsanweisungen müssen am Anfang des Programms auftreten. Sie müssen vor der ersten ausführbaren Anweisung stehen. Vor den Vereinbarungsanweisungen können nur PROGRAM-, SUBROUTINE-, FUNCTION- oder Block DATA-Anweisungen stehen.

4.3.1. Typvereinbarungen

In FORTRAN gilt eine implizite Typzuweisung (Abschnitt 2.8.). Durch eine Typvereinbarung kann dies geändert werden. Die Typvereinbarung hat die allgemeine Form:

t v1,v2,...,vn

wobei t ein Typ aus der in Abschnitt 2.6. angegebenen Menge von Typbezeichnungen ist (BYTE, INTEGER*1,...., REAL * 8). Die vi sind Normen von Variablen, Feldern, Feldelementen oder Funktionen.

Beispiel:
```
INTEGER A,Z
REAL I, J, K
DOUBLE PRECISION R, S, T
```

Nach diesen Typanweisungen sind A und Z vom Typ ganzzahlig, I, J und K vom Typ reell und R, S, T vom Typ doppeltgenau.

4.3.2. DIMENSION- (oder Feld-) Anweisungen

Die DIMENSION-Anweisung dient ausschließlich zur Definition von Feldern. Sie hat die allgemeine Form:

DIMENSION f1, f2, ... fn

Die fi sind Feldbezeichnungen der Art fi (k1), fi (k1, k2) oder fi (k1, k2, k3). fi ist der Name des Feldes und die ki sind die Dimensionen (ganzzahlige Konstante) des Feldes.

Beispiel: FLIP (25), AMAT (5,3), C (3,2,2)

Die Elemente des Feldes werden in der folgenden Weise im Speicher angeordnet: Zuerst wird der linke, dann der mittlere und dann der rechte Index variiert.

z.B.

C(1,1,1), C(2,1,1), C(3,1,1), C(1,2,1), C(2,2,1), C(3,2,1),

C(1,1,2), C(2,1,2), C(3,1,2),

C(1,2,2), C(2,2,2), C(3,2,2).

Ein Element eines mehrdimensionalen Feldes kann auch mit einem Index (Feldnummer) bezeichnet werden. So ist z.B C(1,1,1) = **C(1), C(2,1,1) = C(2)** und **C(3,1,2) = C(9)**
Die Feldnummer I des Elements A(i,j,k) eines durch z.B. DIMENSION A(l,m,n) vereinbarten Feldes ist

I = i+l(j-1)+l-m(k-1)

Bei zweidimensionalen Feldern ist in der Formel k durch 1 zu ersetzen. Ein Feld kann auch in einer Typvereinbarung definiert werden.

z.B.
```
INTEGER A(3)
REAL K(7,9)        ist äquivalent zu
LOGICAL FLIP (25)
```

```
INTEGER A
REAL K
LOGICAL FLIP
DIMENSION A(3)
K(7,9), FLIP (25)
```

Werden Felder durch eine Typenvereinbarung definiert, dann darf die Felddefinition nicht noch in einer DIMENSION-Anweisung auftreten. In Unterprogrammen können die Dimensionen in einer DIMENSION-Anweisung auch ganzzahlige Variable sein (Abschnitt 6.4.).

4.3.3. COMMON-Anweisungen

COMMON-Anweisungen sind Speicherplatzzuweisungen für Variable und Felder. Sie dienen zum Einsparen von Speicherplatz. Verschiedene Programmeinheiten können auf gleiche Speicherbereiche zugreifen.

Allgemeine Form: COMMON /BN1/V1/BN2/V2.../BNn/Vn

BNi sind Blockspeichernamen und die Vi sind eine Liste durch Komma getrennter Variablen-, Feld- oder Feldelementnamen.

Beispiel: COMMON /A/A,B,C,D(4)/BDATA/X,Y,FL(7),Z

Diese Anweisung definiert zwei Speicherblöcke mit den Namen A und BDATA. Block A enthält die Elemente A,B,C,D(1),D(2), D(3), D(4). Der Blockname A hat keine Beziehung zum Variablennamen A. Der Block BDATA enthält die Elemente X,Y,FL(1),...,FL(7),Z.
Fehlt ein Blockname (unbenannter Block), dann sind zwei Schrägstriche zu schreiben. Fehlt BN1, dann können die beiden Schrägstriche weggelassen werden.
Ein Blockname darf in einer COMMON-Anweisung mehrmals vorkommen, die dazugehörigen Listen bilden einen benannten Speicherblock. Eine Feldbezeichnung in einer COMMON-Anweisung definiert dieses Feld. Ein solches Feld darf nicht noch einmal in einer Typvereinbarung oder DIMENSION-Anweisung erklärt werden. Ein schon durch Typvereinbarung oder DIMENSION definiertes Feld wird in der COMMON-Anweisung nur mit dem Feldnamen angegeben.

```
Beispiel:    DIMENSION  A(4),B(2,2)
             COMMON     A/Block/Z(2),C(3)//B
```

Der unbenannte Block enthält dann A(1), A(2), A(3), A (4), B(1,1), B(2,1), B (1,2), B(2,2). Der Speicherblock BLOCK enthält Z(1), Z(2), C(1), C(2), C(3).
Der Name eines Blocks kann in mehr als einer COMMON- Anweisung in verschiedenen Programmeinheiten vorkommen. Es werden jeweils die gleichen Speicherplätze benutzt. Die Länge von Blöcken gleichen Namens kann in den verschiedenen Programmeinheiten unterschiedlich sein. Es muß dann aber die Programmeinheit mit der größten Blocklänge zuerst im Programm auftreten.

```
Beispiel:    Programmeinheit 1:   COMMON /A/Z(3),D(2)/B/Z(3,2,2)
             Programmeinheit 2:   COMMON /A/C(2),B/XYZ/I,J,K
```

4.3.4. EQUIVALENCE-Anweisungen

Durch diese Anweisung kann für mehrere Variable oder Feldelemente der gleiche Speicherplatz benutzt werden.

Allgemeine Form: EQUIVALENCE (V1), (V2),...,(Vn)

Wobei die Vi Listen von durch Komma getrennten Variablen-, Feld- oder Feldelementnamen sind.

Beispiel: EQUIVALENCE (A,B,C)

Bei der Programmabarbeitung wird für A,B und C der gleiche Speicherplatz benutzt.
Alle in Listen auftretenden Feldnamen müssen vorher als Felder **vereinbart** worden sein. Die Indizes eines Feldelements müssen ganzzahlige Konstante sein. Ein Element eines mehrdimensionalen Feldes kann auch durch die Feldnummer des Elementes gekennzeichnet werden.

```
Beispiel:   DIMENSION  A(10), B(10)
            EQUIVALENCE    (A,B)
```

Für die Felder A und B werden die gleichen Speicherplätze benutzt. Durch die EQUIVALENCE-Anweisung können Variable einem COMMON-Bereich zugeornet werden.

```
            COMMON    /X/A,B,C
            EQUIVALENCE  (A,D)
```

A und D benutzen den ersten **Platz** im Block X.
Durch die EQUIVALENCE-Anweisung kann die Größe eines COMMON-**Blocks durch das Anfügen von Elementen an das Ende des Blocks vergrößert werden.**

```
Beispiel:   DIMENSION  A(3),C(5)
            COMMON    /XYZ/A,B
            EQUIVALENCE  (A(2),C(1))
```

Speicherplatz	i	i+1	i+2	i+3	i+4	i+5
XYZ	A(1)	A(2)	A(3)	B		
durch EQUIV.		C(1)	C(2)	C(3)	C(4)	C(5)

Die Größe des Blocks XYZ ist auf 6 erweitert.
Eine Erweiterung nach vorn ist nicht möglich. z.B.

```
            EQUIVALENCE (C(5), A(2))
```

Zwei Namen aus einem COMMON-Block können nicht in einer EQUIVALENCE-Liste auftreten. In einer EQUIVALENCE-Liste dürfen keine formalen Parameter erscheinen.

4.3.5. DATA-Anweisung

Mit dieser Anweisung werden Variablen und Feldern Anfangswerte zugewiesen.

Allgemeine Form: DATA V1/K1/,V2/K2/,...,Vn/Kn/

wobei die Vi-Listen von Variablen-, Feld- oder Feldelementnamen und die Ki entsprechenden Listen von Konstanten sind, die den Elementen der Liste Vi zugewiesen werden.
Wird mehreren aufeinanderfolgenden Elementen aus Vi die gleiche Konstante zugeordnet, kann n*K geschrieben werden.
Die Anzahlen in Vi und Ki müssen übereinstimmen.

```
Beispiel:     DIMENSION  C(7)
              DATA       A,B,C(1),C(3)/14.73,-8.1,2*7.5/
```

Als Anfangswerte können auch hexadezimale und Zeichenkettenkonstante verwendet werden. Sie können Elementen beliebigen Typs zugeordnet werden.

```
Beispiel:     DIMENSION     HAR(2)
              DATA          HAR,B/4HSTOPP,4HOK,7.86/
              !
              REAL          LIT(2)
              LOGICAL       LT,LF
              DIMENSION     H4(2,2),PI3(3)
              DATA    A1,B1,K1,LT,LF,H4(1,1),H4(2,1,,
             1        H4(1,2),H4(2,2),PI3/5.9,2.5E-4,64,
             2        .FALSE.,.TRUE.,1.75E+2,3*1.5,1.,
             3         2.,3.14159/
```

Eine DATA-Liste darf keinen Namen enthalten, der in einer nachfolgenden Vereinbarungsanweisung nochmals auftritt.
Einer Variablen eines unbenannten COMMON-Blocks kann kein Anfangswert zugewiesen werden. Eine Anfangswertzuweisung für Variable benannter COMMON-Blöcke ist nur in BLOCK DATA-Programmeinheiten zulässig (Abschnitt6.6).

4.3.6. EXTERNAL-Anweisung

Die EXTERNAL-Anweisung legt für ein Programm fest, daß die in der Liste auftretenden Namen als Namen externer Unterprogramme zu betrachten sind. Alle Unterprogrammnamen (FUNCTION und SUBROUTINE), die als aktuelle Parameter verwendet werden, müssen in EXTERNAL-Anweisungen enthalten sein.

Allgemeine Form: EXTERNAL name, name, name,..

(Beispiel Abschnitt 6.2).

5. Ein- und Ausgabeanweisungen

FORTRAN bietet eine Reihe von Anweisungen, die die Steuerung und die Bedingungen der Datenübertragung zwischen Speicher und externen Geräten, wie Magnetband, Diskette, Drucker, Lochkarteneinheiten, Tastaturen usw. definieren. Diese Anweisungen lassen sich in folgenden Gruppen zusammenfassen:

1. Formatgebundene READ- und WRITE-Anweisungen;
2. Formatfreie READ- und und WRITE-Anweisungen zur Übertragung binärer Daten;
3. Ein- und Ausgabeanweisungen zur Positionierung und Markierung von Dateien (Files);
4. ENCODE- und DECODE-Anweisungen für die Übertragung von Daten zwischen Speicherplätzen;
5. FORMAT-Anweisungen, die in Verbindung mit formatgebundenen Datensatzübertragungen benutzt werden, um Datenkonvertierung und Informationszusammenstellung zwischen interner Datendarstellung und externen Zeichenkettenformen herzustellen.

5.1. Formatgebundene READ- und WRITE-Anweisungen

Allgemeine Form: READ (u,f, ERR=L1, END=L2) V
WRITE (u,f, ERR=L1, END=L2) V

u ist die Nummer einer physischen und logischen Einheit. u ist entweder eine vorzeichenlose Konstante oder eine ganzzahlige Variable im Bereich von 1 bis 255. Ist u eine ganzzahlige Variable, muß ihr vor der Ausführung von READ oder WRITE ein ganzzahliger Wert zugewiesen worden sein. Die Einheiten 1,3,4 und 5 sind der Konsole zugeordnet, Einheit 2 ist der Drucker. Die Einheiten 6 bis 10 sind den Diskettenfiles zugeordnet.
Diese Einheiten, ebenso die Einheiten 11 bis 255, können durch den Nutzer anders zugeordnet werden (Abschnitt 8.3).

f ist die Anweisungsmarke einer FORMAT-Anweisung, die die Art der Datenkonvertierung beschreibt (Abschnitt 5.4).

L1,L2 sind Anweisungsmarken, zu denen verzweigt wird, wenn ein Einlesefehler auftritt bzw. wenn das Dateiende erreicht wird. Ein Rücksprung zur Unterbrechungsstelle ist nicht möglich. ERR=L1 oder END=L2 können auch fehlen. I/O-Fehler können dann allerdings zu unvorhersehbaren Fehlern führen.

V ist eine Liste von durch Komma getrennten Variablen-, Feld- oder Feldelementnamen. V ist eine Ein-/Ausgabe-Liste.

Eine Ein-/Ausgabe-Liste hat die Form

l1, l2, l3, ..., ln

Die Listenelemente sind Namen von Variablen, Feldnamen oder Feldelementen. Für Felder ist eine Darstellung wie für eine DO-Anweisung zulässig (Implizites-DO).

Beispiele:

Liste	Anordnung	Bemerkung
I,A,B,C(5),F(M)	I,A,B,C(5),F(M)	I,A,B Variable
A,X,C(13)	A,X(1,1),X(2,1), X(1,2),X(2,2),C(13)	X Feld X(2,2)
(X(I), I=1,4)	X(1),X(2),X(3),X(4)	
(Q(J),R(J),J=1,2)	Q(1),R(1),Q(2),R(2)	
(R(3),T(I),I=1,3)	R(3),T(1),R(3),T(2), R(3),T(3)	
((A(I,J),I=3,5), J=1,9,4)	A(3,1),A(4,1),A(5,1)	
	A(3,5),A(4,5),A(5,5)	
	A(3,9),A(4,9),A(5,9)	
J,(I,A(I),I=1,J,2)	5,1,A(1),3,A(3),5,A(5)	J = 5

Konstante dürfen nicht als Listenelemente auftreten. Bei Eingabelisten dürfen die impliziten-DO-Elemente k, m_1, m_2, m_3 (Abschnitt 4.2.3) nicht als Listenelemente innerhalb der Klammer auftreten.

Beispiel: (I,J,A(I), I = 1,J,2) ist nicht zugelassen;
I,J,(A(I),I = 1,J,2) ist zugelassen.

Bemerkungen zu READ (u,f) V.

Mit jeder Abarbeitung der READ-Anweisung wird ein neuer Datensatz vom Eingabefile (Eingabedatei) gelesen.
Die Anzahl der Datensätze, die gelesen werden, wird bestimmt durch die Liste V und durch die FORMAT-Festlegung.
Die Anzahl der Elemente der Liste V ist nicht begrenzt. Die Listenelemente können verschiedenen Typs sein.
Es werden nur soviel Daten eingelesen, wie Elemente in der Liste V enthalten sind. Die überzähligen Daten werden ignoriert.

Bemerkungen zu WRITE (u,f) V

Mit einer WRITE-Anweisung können mehrere Daten ausgegeben werden. Die Daten können unterschiedliche Typen haben. Die Anzahl der Daten wird durch die Liste V festgelegt.
Aufeinanderfolgende Daten werden ausgegeben bis die durch die Liste V spezifizierten Daten ausgeschöpft sind.

Füllen die auszugebenden Daten das angegebene Format nicht aus, werden Leerzeichen aufgefüllt.

Für die Ein- und Ausgabe alphanumerischer Informationen wird keine Liste benötigt.

Durch READ (I, 25)

25 FORMAT (10HABCDEFGHIJ)

werden die nächsten 10 Zeichen vom Eingabegerät I gelesen und an die Stelle der 10 Zeichen der FORMAT-Anweisung eingesetzt.

Durch WRITE (2,26)

26 FORMAT (12HABCDEFGHIJ)

wird der Text ABCDEFGHIJ ausgegeben.

5.2 Formatfreie READ- und WRITE-Anweisungen

Allgemeine Form: READ (u, ERR = L1, END = L2) V

WRITE (u, ERR = L1, END = L2) V

mit den in 5.1 angegebenen Bedeutungen. Formatangaben fehlen. Mit diesen Anweisungen werden Übertragungen binärer Daten durchgeführt. Der übertragene Datenumfang entspricht der Anzahl der Variablen der Liste V.
Die Anzahl der Variablen in READ (u) V darf nicht größer sein als die Datensatzlänge.
WRITE (u) V schreibt einen logischen Datensatz.
Ein logischer Datensatz kann aus mehr als einem physischen Datensatz bestehen.

5.3. ENCODE- und DECODE-Anweisungen

Diese beiden Anweisungen übertragen entsprechend der Formatspezifikation von einem Bereich des Speichers in einen anderen. DECODE wandelt ASCII-Format in das angegebene Format um, während ENCODE Daten des angegebenen Formats in das ASCII-Format wandelt.
Die Anweisungen haben die Form:

ENCODE (A,f) V

DECODE (A,f) V

wobei A ein Feldname, f eine FORMAT-Anweisungsmarke und V eine Ein-/Ausgabeliste ist.

Das Feld A muß groß genug sein, damit es alle zu verarbeitenden Daten enthalten kann. Es gibt keine Überlaufprüfung. Ein Überlauf bei einer ENCODE-Anweisung überschreibt Daten, die nach dem Feld kommen. Eine DECODE-Anweisung verarbeitet bei Überlauf auch noch Daten nach dem Feld.

5.4. FORMAT-Anweisungen

Eine FORMAT-Anweisung dient der Ein- und Ausgabe von Daten in aufbereiteter Form. Sie enthält alle notwendigen Informationen, um die auf dem Eingabemedium gespeicherten Daten bei der Eingabe richtig zu interpretieren und intern gespeicherte Daten in einer gewünschten Form auszugeben.

Eine FORMAT-Anweisung hat die Form:

FORMAT (Format, Format, ...)

wobei Format eines der im folgenden beschriebenen Formate ist. Sonderformen werden im Abschnitt 5.5 erläutert.

Eine FORMAT-Anweisung kann an beliebiger Stelle im Programmkörper stehen. Innerhalb der Programmeinheit kann man sich in READ/WRITE-Anweisungen auf diese FORMAT-Anweisung beziehen. Sie muß deshalb eine Marke haben.

Es gibt Formate: I,E,D,F,L,G,A,X. Zur Formatangabe kommt noch die Angabe der Feldwerte (w) und der Dezimalstellen (d). Die Bedingungen für w und d werden bei der Beschreibung der einzelnen Formate angegeben. Bei den Formaten E,D,F und G kann ein Skalenfaktor nP vor die Formatbezeichnung gesetzt werden, (z.B. 4PE10,5). Die Wirkung des Skalenfaktors wird bei den Formatbeschreibungen behandelt (Abschnitt 5.4.5).

5.4.1 I-Format

Iw mit $1 \leq w \leq 255$

Das I-Format dient zur Übertragung ganzzahliger Daten.

Eingabe: Das Vorzeichen + kann entfallen. Führende Leerzeichen sind bedeutungslos. Leerzeichen im Datenfeld werden als Ø interpretiert.

Beispiele:	FORMAT	Eingabe	interner Wert
	I4	␣124	124
	I4	-124	-124
	I7	␣␣67328	6732Ø
	I4	1␣2␣	1Ø2Ø

Ausgabe: Die ganze Zahl (ev. mit Vorzeichen Minus) wird rechtsbündig in das Datenfeld übertragen. Nach links werden (falls nötig) Leerzeichen aufgefüllt. Die Anzahl der Zeichen, die zur Darstellung der Zahl notwendig ist (einschließlich Minuszeichen, falls die Zahl negativ ist), darf die angegebene Feldweite w nicht überschreiten. Falls dies nicht beachtet wurde, wird das Datenfeld mit * gekennzeichnet.

Beispiele:	FORMAT	interner Wert	Ausgabe
	I6	+281	␣␣␣281
	I6	-23261	-23261
	I3	126	126
	I4	-226	-226
	I3	1234	*34

Anweisungsbeispiele: Leerzeichen ist durch Unterstrich gekennzeichnet.

a) Datenanordnung:

```
139__-15__18__7 ; READ (u,f) K1,K2,K3,K4

f FORMAT (I3,I5,I4,I3) ergibt Ki=139,-15,18,7
f FORMAT (I4,I5,I4,I2) ergibt Ki=139Ø,-15Ø,18Ø,7
```

b) Daten: Ki=-601,2Ø,75,1Ø27 WRITE (u,f) K1,K2,K3,K4
f FORMAT (I4,I4,I4,I6) ergibt -6Ø1__2Ø__75__1Ø27
f FORMAT (I5,I6,I2,I3) ergibt _-601____2Ø75*27

5.4.2 E-, D-, F-Format

Ew.d Dw.d Fw.d $1 \leq w \leq 255$, $\emptyset \leq d \leq w$

Diese Formate dienen zur Übertragung reeller Daten einfacher und doppelter Genauigkeit. Der Typ des zugehörigen Elementes der E/A-Liste entscheidet, ob die Daten als reelle Zahlen einfacher oder doppelter Genauigkeit übertragen werden. Die Formate haben daraufhin keinen Einfluß.

Eingabe im E-, D- und F-Format: Die Daten können relativ beliebig formuliert werden. Führende Leerzeichen werden ignoriert. Enthält das Datenfeld einen Dezimalpunkt, so bestimmt dieser die Anzahl der Dezimalstellen. Enthält das Datenfeld keinen Exponenten (E oder D) und keinen Dezimalpunkt, werden die letzten (rechten) d Ziffern oder Leerzeichen des Datenfeldes als gebrochener Teil und die davorstehenden Ziffern oder Leerzeichen als ganzer Teil der Zahl gewertet. Enthält das Datenfeld einen Exponenten und keinen Dezimalpunkt, so werden die letzten d Zeichen im Datenfeld vor E oder D als gebrochener Teil der Zahl gewertet. Der eingelesene Wert ergibt sich aus diesem, multipliziert mit dem Wert des Exponenten. Ein Exponent ohne Ziffern davor ist nicht erlaubt. Sind im Datenfeld nur Leerzeichen, ist die Zahl Ø.

Beispiel: Format E4.Ø

Datenfeld	␣3ØØ	3.33	3E-3	.3D3
Zahl	Ø.3*10**3	Ø.333*10**1	Ø.3*10**(-2)	Ø.3*10**3

Format E1Ø.5

Datenfeld	␣+15432␣EØ	-54321E+2Ø
Zahl	+Ø.15432Ø*10**1	-Ø.54321*10**2Ø

Ausgabe im F-Format: Bei der Ausgabe im F-Format werden insgesamt w Zeichen einschließlich Minuszeichen und Dezimalpunkt ausgegeben. Hinter dem Dezimalpunkt stehen d Zeichen.

Beispiel:

	Format F5.Ø			Format F7.2	
Zahl	1234.567	Ø.12	2Ø.3	1234.567	Ø.12
Datenfeld	1235.	␣␣␣␣Ø.	␣␣2Ø.	1234.57	␣␣␣Ø.12

Ausgabe im E- und D-Format: Die Daten werden in der Form VØ.zzzz...zEVzz ausgegeben. V kann ein Minuszeichen oder Leerzeichen sein. Ein Pluszeichen wird nicht angegeben. Die Anzahl der Ziffern zwischen Dezimalpunkt und E ist gleich d.

Beispiel:

Format	E1Ø.4	E7.3	E13.5	E8.2
Zahl	-1234567.	56.93	13.31	Ø.Ø
Datenfeld	-.1235E6Ø7	✱Ø.569E	66Ø.1331ØE6Ø2	Ø.Ø66666

5.4.3 L-Format

Lw , $1 \leq w \leq 255$

Das L-Format dient zur Übertragung logischer Daten.

Eingabe: Der Variablen wird der Wert TRUE bzw. FALSE zugeordnet, wenn im Datenfeld das erste vom Leerzeichen verschiedene Zeichen ein T bzw. ein F ist. Hinter T bzw. F können beliebige Zeichen folgen.

Ausgabe: Ist der Wert des Listenelements Ø, dann wird ein F, ist er ≠Ø, dann wird ein T rechtsbündig in das Datenfeld übertragen. Davor erscheinen w-1 Leerzeichen.

5.4.4 G-Format

Gw.s , $1 \leq w \leq 255$, $Ø \leq s \leq w$

Das G-Format wird zur Übertragung ganzzahlig reeller und logischer Daten verwendet. Durch den Typ des E/A-Listenelements wird bestimmt, wie die Daten interpretiert bzw. dargestellt werden.

Für ganzzahlige und logische Daten ist s bedeutungslos und kann weggelassen werden. Für reelle Daten hat s bei der Eingabe die gleiche Bedeutung wie d im D-, E- und F-Format. Die Bedeutung bei der Ausgabe wird im folgenden erläutert.

Eingabe: Abhängig vom Typ des E/A-Listenelements ist Gw.s gleichwertig mit

Iw bei INTEGER , Lw bei LOGICAL und

Dw.d, Ew.d, Fw.d bei REAL

Ausgabe: Die Ausgabe erfolgt entsprechend dem Typ und dem Wert (Betrag) des E/A-Listenelements wie im Format

Iw	bei INTEGER
Lw	bei LOGICAL
Ew.s	bei REALx4 und Wert < Ø.1 oder Wert $\geq 1Ø^{✱✱s}$
Dw.s	bei REALx8 " " " " " "
F(w-4).(s-K),4x	bei REAL und $1Ø^{K-1} \leq$ Wert $< 1Ø^{K}$; $Ø \leq K \leq S$

Beispiel: Variable vom Typ REAL✱4

Format	G8.3	G8.3	G11.3	G9.3
Zahl	Ø.Ø123	123.Ø	1Ø23.4	123.5
Datenfeld	.123E6-1	123.6666	66Ø.1Ø2E6Ø4	6123.6666

5.4.5 Wirkung des Skalenfaktors nP

nP $-127 \leq n \leq 127$

n wird automatisch Ø gesetzt bei jedem formatgebundenen E/A-Aufruf. Nach dem Auftreten eines Skalenfaktors haben alle nachfolgenden Datenfeldbeschreibungen (in einem Format) denselben Skalenfaktor. Durch die Angabe von ØP wird die Wirkung aufgehoben. Im folgenden bezeichnet EZ und IZ eine externe bzw. interne Zahl.

Datenfeld-beschreibung	Eingabe Exponent	Eingabe kein Exponent	Ausgabe
nPFw.d	Skalenfaktor wirkungslos	$IZ = EZ.10^{-n}$	$EZ = IZ.10^{n}$
nPEW.d			Die Mantisse wird mit 10^{n} multipliziert und vom Exponenten wird n abgezogen
nPDw.d			
nPGw.d			wenn Ew.d zutrifft, Wirkung wie oben, sonst wirkungslos.

5.4.6 A-Format

Aw $1 \leq w \leq 255$

Mit diesem Format werden Literale (Zeichenketten) zwischen einem externen Speichermedium und Variablen beliebigen Typs übertragen. Die Zahl der Zeichen, die in den Speichereinheiten des E/A-Listenelements dargestellt werden können, ist gleich der Typlänge g (Abschnitt 2.6).

Eingabe: Ist $w \leq g$, dann werden die w Zeichen des Datenfeldes in die Speichereinheit übertragen.
Ist $w < g$, werden die w Zeichen des Datenfeldes linksbündig in die Speichereinheit übertragen und die rechten (g-w) Bytes der Speichereinheit mit Leerzeichen gefüllt.
Ist $w > g$, werden die ersten links im Datenfeld stehenden (w-g) Zeichen übergangen und nur die nächsten g Zeichen übertragen.

Ausgabe: Bei w = g werden die g Zeichen aus der Speichereinheit in das Datenfeld übertragen.
Ist $w > g$, werden (w-g) Leerzeichen, gefolgt von den g Zeichen in das Datenfeld übertragen.
Ist $w < g$, werden nur die ersten in den linken Bytes gespeicherten w Zeichen in das Datenfeld übertragen.

Beispiel:

```
      DIMENSION KARTE (2Ø)
      READ (1,1Ø) KARTE
1Ø FORMAT (A8Ø)
      READ (1,2Ø) KARTE
2Ø FORMAT (4ØA2)
```

Durch die erste READ-Anweisung werden aus den nächsten 20

Datensätzen der Datei 1 die in den Positionen 79-80 stehenden Informationen in das Feld KARTE übertragen.
Mit der zweiten READ-Anweisung wird der Inhalt der Datei 1 nach KARTE übertragen (4Øx2 Zeichen).

5.4.7 Literalformate

wHc1....cw oder 'c1.....cw' , $1 \leq w \leq 255$

Die ci können beliebige Zeichen sein. Beide Formate dienen zur Übertragung von Literalen zwischen externen Speichermedien und dem im FORMAT stehenden Literal c1....cw. In der Form 'c1...cw' auftretende Apostrophe müssen doppelt geschrieben werden.

Eingabe: Die w Zeichen des Datenfeldes werden ohne Konvertierung in das Literal der FORMAT-Anweisung übertragen. Das dort stehende Literal wird überschrieben.

Ausgabe: Die zum Zeitpunkt der Ausgabe im FORMAT stehende Zeichenkette der Länge w wird in das Datenfeld des Satzes übertragen.

Beispiel:
```
      READ (1,1Ø)
1Ø FORMAT ('␣EIN␣MANN␣steht␣VOR␣DER␣TUER')
      WRITE (2,1Ø)
```

Befindet sich auf der Datei 1 der Datensatz

```
1EIN␣BAUM␣STEHT␣VOR␣DEM␣HAUS
```

dann wird auf die Datei 3 (z.B. Drucker) als Position 1 der Satz EIN␣BAUM␣STEHT␣VOR␣DEM␣HAUS ausgegeben. Die 1 im Ausgabesatz wird zur Vorschubsteuerung benutzt (Abschnitt 5.5).

5.4.8 X-Format

wX $1 \leq w \leq 255$

Eingabe: w Zeichen des Datenfeldes werden übergangen.

Ausgabe: In das Datenfeld werden w Leerzeichen übertragen.
X entspricht keinem Element der E/A-Liste.

5.5 Spezielle Steuereigenschaften von FORMAT-Anweisungen

Im folgenden werden Sonderformen der Anordnung der Formate in der Formatliste einer FORMAT-Anweisung beschrieben.

FORMAT (Format1, Format2, ..., Formatn)

Die Formate i können mit einem Wiederholfaktor versehen werden. Durch Klammerung können aus Formaten Formatgruppen gebildet werden. Diese Gruppen können ebenfalls Wiederholfaktoren haben. z.B.:

2(2I1,E1Ø.3) ist äquivalent zu 2I1, E1Ø.3, 2I1, E1Ø.3

Tritt in einer Gruppe mit Wiederholfaktor ein Literalformat auf, dann gilt die Äquivalenz nicht. z.B.

2('A',I1) nicht äquivalent zu 'A', I1, 'A', I1

Eine geklammerte Formatliste darf in höchstens einer Formatgruppe enthalten sein. z.B.

```
1 FORMAT (I1Ø,'TITEL', 12('NAME', I8, 4(I5,F1Ø.2,'M')))

2 FORMAT (I1, 4(I2,I3,3(I4,I5,2(I6,I7),I8)))
```

1 ist eine erlaubte FORMAT-Anweisung. Die Formatgruppen in 2 sind zu tief geschachtelt.

Die letzte äußere Formatgruppe hat in der Formatsteuerung noch eine weitere Funktion. Enthält die Formatliste weniger Formate als die E/A-Liste fordert, so wird das Abarbeiten mit der am weitesten rechts stehenden Formatgruppe wiederholt unter Einbeziehung des Wiederholfaktors. Durch diese Art der Wiederholung wird das Verarbeiten des momentanen Datensatzes (Record) beendet und die Verarbeitung eines neuen Datensatzes begonnen.

z.B.

```
a) DIMENSION A(1ØØ)
   READ (3,13) A
13 FORMAT (5F7.3)
```

In diesem Beispiel werden jeweils die ersten 5 Daten von einem der 20 Datensätze eingelesen und den A(i) zugeordnet.

```
b) WRITE (2,12)E,F,K,L,M,
   1 KK,LL,MM,K3,L3,M3
12 FORMAT (2F9.4,(3I7))
```

Es werden drei Datensätze geschrieben
Datensatz 1: E,F,K,L,M
Datensatz 2: KK,LL,MM
Datensatz 3: K3,L3,M3

Zwei in einer Formatliste benachbarte Formate müssen entweder durch ein Komma oder durch ein oder mehrere Schrägstriche getrennt sein.

Ein Schrägstrich bedeutet gleichzeitig den Übergang zu einem neuen Datensatz. Der Rest eines Eingabedatensatzes wird übergangen, der Rest eines Ausgabedatensatzes wird mit Leerzeichen gefüllt

```
z.B. FORMAT (I1,I2,I3/    / 3A4,L7/2X,F5.Ø)
              |  1.   | 2. |   3.  |  4.   | Datensatz.
              |  5.   |usw.
```

Eingabebeispiel:

```
   DIMENSION B(1Ø)
   READ (4,7) B
 7 FORMAT (F1Ø.2/F1Ø.2//(8F1Ø.2)
```

B(1) und B(2) bekommen ihre Werte von den ersten Datenfeldern des 1. und 2. Datensatzes, B(3/bis B(10) bekommen ihre Werte vom 5. Datensatz. Der 3. und 4. Datensatz wird übergangen.

Ausgabebeispiel:

```
   DIMENSION A(100),J(20)
   WRITE (2,8) J,A
 8 FORMAT (10I7/10I7/50F7.3/
          50F.7.3)
```

Es werden 4 Datensätze erzeugt:
1. J(1) bis J(10),
2. J(11)bis J(20),
3. A(1) bis A(50),
4. A(51)bis A(100).

Das erste Zeichen jedes formatgebundenen Ausgabedatensatzes wird zur Druckersteuerung benutzt und deshalb nicht gedruckt. In Abhängigkeit vom ersten Zeichen geschieht bei Zeichen:

+	keine Bewegung
Ø	2 Zeilen weiterschalten
1	Einfügen eines Form Feed
sonst	1 Zeilenvorschub.

Formatfestlegung in Feldern: Der FORMAT-Bezug f in formatgebundenen READ-/WRITE-Anweisungen kann auch ein Feldname sein. Dann muß zur Ausführungszeit der READ-/WRITE-Anweisung der Anfang des Feldes eine gültige, in Klammern eingeschlossene Formatliste enthalten.

Beispiel: a)
```
DIMENSION A(3),B(3),M(3)
DATA A/'(3F1','Ø.3,','4I6)'/
READ (6,A) B,M
```

b)
```
DIMENSION IA(4),B(3),M(3)
READ (7,15) IA
15 FORMAT (4A2)
READ (7,IA) B,M
```

a) Das Feld A enthält durch Dateninitialisierung die Zeichenkette (3F1Ø.3,4I6),

b) In das Feld IA wird durch Einlesen im A-Format die Zeichenkette (3F1Ø.3,4I6) eingespeichert.

5.6 Diskfile-Zugriff

FORTRAN bietet die Möglichkeit des Zugriffes zu Diskfiles durch FORTRAN-Programme. Die LUNs 6 bis 1Ø sind standardmäßig den Diskfiles zugeordnet. Ein READ oder WRITE zu einer LUN eröffnet automatisch das File für Eingabe bzw. Ausgabe, wenn sie nicht schon eröffnet ist. Das File bleibt eröffnet, bis es durch eine ENDFILE-Anweisung oder das Programmende geschlossen wird. Ein File, das durch eine READ- oder WRITE-Anweisung eröffnet wird, bekommt einen Standardnamen, in dem die LUN eingefügt ist:

FORTØ6.DAT, FORTØ7.DAT, ..., FORT1Ø.DAT

Ein File kann auch durch eine OPEN-SUBROUTINE eröffnet werden. Die Form des Aufrufs lautet:

CALL OPEN (LUN, Fname, Laufwerk)

mit

LUN = Nummer einer logischen Einheit, die dem File zugeordnet wird. LUN ist eine ganzzahlige Konstante oder Variable mit einem Wert zwischen 1 und 1Ø. Es können auch die LUNs 1 bis 5 verwendet werden (Logical unit number).

Fname = kennzeichnet den Namen, den das Ø5 dem File zuordnet. Der Name muß mit einem Leerzeichen abgeschlossen werden und dann insgesamt in Apostrophe eingeschlossen sein.

Der Name kann bis zu 32 Zeichen lang sein. Z.B. 'FILENAME5'. Dieses File kann vom Ø5 über FILENAME erreicht werden.

Laufwerk = ist die Nummer des Diskettenlaufwerks, auf dem sich das File befindet oder später befinden soll. Sie ist in der Form 'Ø', '1', usw. anzugeben oder mit '*'. Laufwerk soll ein INTEGER*1-Feld oder einfache Variable sein oder eine Literalkonstante.

Mit der OPEN-SUBROUTINE kann das Programm einer LUN einen Filenamen und ein Laufwerk zuordnen.
Ein OPEN eines nichtvorhandenen Files erzeugt ein Nullfile mit entsprechendem Namen. Ein OPEN eines vorhandenen Files mit anschließender Ausgabe zerstört das vorhandene File.

Ein OPEN eines vorhandenen Files gefolgt von einer Eingabe gestattet den Zugriff auf den Inhalt des Files.

Die ENDFILE- und REWIND-Anweisungen erlauben weitere Programmsteuerungen von Diskfiles. Die Form der Anweisung ist:

```
ENDFILE L oder REWIND L
```

wobei L ein LUN ist. ENDFILE L schließt das dem LUN L zugeordnete File. REWIND L schließt das File ab und eröffnet es wieder.

Es kann weiter auf einzelne Records zugegriffen werden. Die Recordnummer wird in der READ- oder WRITE-Anweisung mit REC = n angegeben. Z.B.

```
I = 1Ø
WRITE (6,2Ø,REC=I,ERR=5Ø) X,Y,Z
```

Damit wird der Datensatz (Record) 1Ø auf das LUN 6 geschrieben. Existiert Record 1Ø schon, wird er überschrieben. Existiert noch kein Record 1Ø, wird das File erweitert. Der Versuch, einen nichtexistierenden Record zu lesen, ergibt einen I/O-Fehler.
Die Recordlänge wird mit 128 Bytes (1 Sektor) festgelegt.
Ein File, von dem recordweise gelesen werden soll, sollte über FORTRAN-Zugriffsanweisungen erzeugt worden sein. Sie können mit binären oder formatgebundenen WRITE-Anweisungen erzeugt werden. Wenn durch die WRITE-Anweisung nicht ausreichend Daten in den Record übertragen werden, um den Record zu füllen (128 Bytes), dann wird der Record mit Nullen (Null-Zeichen) gefüllt.

5.7 I/O-Interface

I/O-Operationen werden mit Hilfe einer Tabelle auf die Treiberroutine für die passende logische Einheit verteilt. Die Verteilertabelle ist %LUNTB (Logical Unit Number Table). Sie enthält eine 2-Byte-Treiberadresse für jede mögliche LUN.
Am Anfang der Tabelle steht die maximale Anzahl der LUNs plus 1.
Das Standard-Laufzeitpaket ist für 10 LUNS (1-1Ø) ausgelegt, die alle dem TTY entsprechen.
Sie können vom Anwender neu definiert und die Anzahl erhöht werden durch Änderung der Plätze in LUNTB. Das Laufzeitsystem benutzt LUN 3 für Fehler- und andere Meldungen. LUN 3 muß deshalb der Konsole entsprechen.

Die Gerätetreiber enthalten lokale Verteilertabellen für die sieben möglichen Operationen pro Gerät (Device):

1) Formatgebundenes Lesen,
2) Formatgebundenes Schreiben,
3) Binäres Lesen,
4) Binäres Schreiben,
5) Rewind,
6) Backspace,
7) Endfile.

Die Startadressen dieser sieben Routinen sind am Anfang des Treibers in der oben angegebenen Reihenfolge plaziert. Die Adresse aus ¤LUNTB zeigt auf diese lokale Tabelle und das Laufzeitsystem indiziert die Adresse, um die Adresse der passenden Routine zu bekommen, mit der die geforderte I/O-Operation durchgeführt werden kann.

Die folgenden Vereinbarungen gelten für die individuellen I/O-Routinen:

1. Der Platz ¤BF enthält die Datenpuffer-Adresse für READs und WRITEs.

2. Bei einem WRITE enthält der Platz ¤BL die Anzahl der zu schreibenden Bytes.

3. Bei einem READ enthält ¤BL die Anzahl der gelesenen Bytes.

4. Alle I/O-Operationen setzen vor der Beendigung die Flags:

   a) CY = 1 , Z beliebig : I/O-Fehler;

   b) CY = 0 , Z = 0 : End-of-file erreicht;

   c) CY = 0 , Z = 1 : normaler Abschluß.

Das Laufzeitsystem prüft die Flags nach dem Treiberaufruf. Ergibt die Prüfung eine nichtnormale Bedingung, wird an den durch "ERR=" oder "END=" definierten Marken fortgesetzt. Fehlt ERR oder END, kann sich ein fataler Fehler ergeben.

5. Die globale Routine ¤IOERR druckt eine "ILLEGAL I/O OPERATION"-Meldung (nicht fatal). Diese Routine kann benutzt werden, wenn es für ein spezielles Gerät nichterlaubte Routinen gibt. Bemerkung: Der I/O-Puffer hat eine feste maximale Länge von 132 Bytes. Wenn ein Treiber Eingabeoperationen zuläßt, die über das Ende des Puffers hinausschreiben, kann es zu unvorhersehbaren Fehlern kommen.

6. Unterprogramme und Funktionen

FORTRAN bietet vielfältige Möglichkeiten zur Definition und Nutzung oft benutzter Programmeinheiten.
Man unterscheidet:

Anweisungsfunktionen,
FUNCTION-Unterprogramme,
SUBROUTINE-Unterprogramme,
Standard-Funktionen und -Unterprogramme.

Jede dieser Prozeduren wird unter einem Namen geführt, der aus max. 6 alphanumerischen Zeichen besteht, wobei das erste

Zeichen ein Buchstabe sein muß.

6.1 Anweisungsfunktionen

Anweisungsfunktionen werden durch eine einzige arithmetische oder logische Ergibtanweisung innerhalb einer Programmeinheit definiert. Sie können nur in dieser Programmeinheit verwendet werden. Zu gleichnamigen Größen in anderen Programmeinheiten besteht keine Beziehung.

Allgemeine Form: fname (for.p,for.p,...) = ausdruck

mit fname = Name der Funktion, for.p = formaler Parameter

und ausdruck = arithmetischer oder logischer Ausdruck.

Die Definitionen von Anweisungsfunktionen müssen nach allen Vereinbarungs- und vor allen ausführbaren Anweisungen stehen.

Der Typ des Anweisungsfunktionsnamens und der formalen Parameter muß explizit durch Typanweisungen definiert werden oder wird durch die Standard-Typfestlegung gegeben. Als Parameter sind auch Feldelemente zugelassen. Der Ausdruck muß den Regeln in Abschnitt 3. entsprechen. Enthält der Ausdruck den Aufruf einer Anweisungsfunktion, muß diese vorher definiert worden sein. Der Name der Anweisungsfunktion selbst darf nicht im Ausdruck vorkommen. Ein im Ausdruck auftretender Funktionsaufruf, der einen der formalen Parameter als aktuellen Parameter hat, darf dessen Wert (ev. durch Nebeneffekte) nicht ändern. Tritt ein Anweisungsfunktionsname in einer Typanweisung auf, darf ihm kein Anfangswert zugewiesen werden. Er darf auch nicht als Feld vereinbart werden.

Der Aufruf der Anweisungsfunktion hat die Form:

fname (akt.p,akt.p,...)

Anzahl und Typ formaler und aktueller Parametern müssen übereinstimmen.

```
Beispiel    FUNC1(A,B,C,D) = ((A+B)**C)/D
   Aufruf:  A12 = A1 + FUNC1(X,Y,27.,C7)
```

6.2 FUNCTION-Unterprogramme

Ein FUNCTION-Unterprogramm ist eine selbständige Programmeinheit. Die in ihr auftretenden Namen von Variablen, Feldern und Anweisungsfunktionen sowie Marken haben keine Beziehung zu gleichen Namen in anderen Programmeinheiten.

```
Allgemeine Form: t FUNCTION fname (formaler Parameter,....,
                                   f.P.)
                 Programmkörper (mit RETURN)
                 END
```

t ist eine Typbezeichnung (INTEGER,REAL usw). Fehlt t, dann wird der Typ entsprechend dem Funktionsnamen fname durch die implizite Standardzuweisung vorgenommen. Ein FUNCTION-Unterprogramm liefert einen Wert. Der Funktionsname muß wenigstens einmal auf der linken Seite einer Ergibtanweisung oder als

Element einer Eingabeliste einer READ-Anweisung auftreten. Weitere Werte können zurückgegeben werden, indem formalen Parametern Werte zugewiesen werden. Formale Parameter können sein: Variablennamen, Feldnamen oder formale Namen von SUBROUTINE- oder Namen anderer FUNCTION-Unterprogramme. Es muß wenigstens ein formaler Parameter auftreten.
Die Namen in der Liste der formalen Parameter dürfen nicht in COMMON-, EQUIVALENCE- oder DATA-Anweisungen des FUNCTION-Unterprogramms vorkommen. Ein FUNCTION-Unterprogramm kann jede FORTRAN-Anweisung enthalten, außer BLOCK DATA-, SUBROUTINE-, andere FUNCTION-Anweisungen oder eine Anweisung, die sich auf das definierte FUNCTION-Unterprogramm bezieht. Das logische Ende eines FUNCTION-Unterprogramms ist RETURN. Es muß wenigstens ein RETURN auftreten. Das physische Ende ist END.

Beispiele a)
```
      FUNCTION  Z7(A,B,C)
      Z7 = 5.*(A-B)+SQRT(C)
      B = B+Z7
      RETURN
      END
```

Z7 und B sind dann im aufrufenden Programm verfügbar, der Wert von B dann unter dem entsprechenden aktuellen Namen.

b)
```
      FUNCTION  SUM(BARY,I,J)
      DIMENSION BARY(10,20)
      SUM = 0.0
      DO 8 K = 1,I
      DO 8 M = 1,J
    8 SUM = SUM + BARY(K,M)
      RETURN
      END
```

Aufrufe: a)
```
      Z10 = FT1 + Z7(D,T3,RHO)
```
b)
```
      DIMENSION DAT (5,5)
      S1 = TOT1 + SUM(DAT,5,5)
```

Ist ein aktueller Parameter ein Unterprogrammname, muß dieser Name von den Namen üblicher Variabler dadurch unterschieden werden, daß er in einer EXTERNAL-Anweisung erscheint. Die entsprechenden formalen Parameter müssen im FUNCTION-Up in Up-Bezügen genutzt werden.

Beispiel
```
      FUNCTION PI(X,FUN)
      PI = X**2+FUN(X)
      RETURN
      END
```

Aufrufendes Programm:
```
      EXTERNAL COS,EXP
      Z = PI(Y,COS) + PI(Y,EXP)
```

6.3 SUBROUTINE-Unterprogramme

Allgemeine Form:

```
SUBROUTINE sname (formaler Parameter,...,f.P.)
    Programmkörper (mit RETURN)
    END
```

sname ist der Name des SUBROUTINE-Unterprogramms. Der Name besitzt keinen Typ. Der SUBROUTINE-Name darf im Programmkörper nicht vorkommen.

Werden von den formalen Parametern Werte dargestellt, die an das rufende Programm zurückgegeben werden, müssen diese formalen Parameter auf der linken Seite einer Ergibtanweisung, in der Eingabeliste einer READ-Anweisung oder als ein Parameter in einem Unterprogrammbezug erscheinen. Im allgemeinen gilt für SUBROUTINE-Unterprogramme das im Abschnitt 6.2 über FUNCTION-Unterprogramm Gesagte.
Ein SUBROUTINE-Unterprogramm wird mit der CALL-Anweisung aufgerufen.

```
CALL sname (akt.Param.,akt. Param.,...)
```

Beispiel a)

```
SUBROUTINE MATRIX(A,X,Y,N)
DIMENSION A(50,50),X(50),Y(50)
DO 2 I = 1,N
S = 0.0
DO 1 J = 1,N
1 S = S+A(I,J)*X(J)
2 Y(I) = S
RETURN
END
```

Aufruf:

```
CALL MATRIX(C,D,F,25)
```

C,D,F müssen als Felder vereinbart sein.

Der Wert der Variablen S ist im rufenden Programm nicht verfügbar.

Beispiel b)

```
SUBROUTINE SUBR(A,B,C)
READ(3,7)B
A = B**C
RETURN
7 FORMAT (F9.2)
END
```

Aufruf:

```
CALL SUBR(Z9,B7,R1)
```

Z9 und B7 sind im aufrufenden Programm verfügbar.

6.4 Verarbeitung von Feldern in Unterprogrammen (UP)

Soll einem Unterprogramm ein Feldname übergeben werden, dann muß das UP die zu diesem Feld gehörende Dimensionsfestlegung enthalten. Wenn ein formaler Parameter ein Feld- oder Feldelementname ist, dann muß das UP Feldvereinbarungen enthalten.
Variable Dimensionen können nur benutzt werden, wenn Feldname und alle Dimensionen formale Parameter sind. Im gerufenen Programm dürfen die variablen Dimensionen nicht geändert werden.

Beispiel
```
SUBROUTINE SUBR(X,I,J,K)
DIMENSION
:
RETURN
END
```

Aufruf
```
DIMENSION A(3,4,5)
:
CALL SUBR (A,3,4,5)
:
```

6.5 Standard-Unterprogramme

Standardfunktionen werden mit einem Funktionsaufruf aufgerufen. Anzahl und Typ der Parameter, Typ des Ergebnisses (Funktionswert) und weitere Bedingungen werden in der Tabelle im Abschnitt 1.1 angegeben. Der Typ einer Standardfunktion braucht nicht explizit vereinbart zu werden. Die Namen der Standardfunktionen sollten nicht für andere Zwecke benutzt werden. Es werden out-of-line- und In-line-Standardfunktionen unterschieden.
Out-of-line-Standardfunktionen (OS) ähneln FUNCTION-Unterprogrammen. Der Code zu ihrer Realisierung wird nur einmal in ein ausführbares Programm eingefügt. Der Name einer solchen Funktion ist der Name einer externen Funktion. Er kann als aktueller Parameter in UP-Aufrufen verwendet werden. Er muß dann in einer EXTERNAL-Anweisung enthalten sein.
In-line-Standardfunktionen (IS) werden realisiert, indem ihr Code bei jedem Aufruf in das Objektprogramm eingefügt wird. Der Name einer solchen Funktion ist kein Name einer externen Funktion, er kann nicht aktueller Parameter sein.

Im U880-FORTRAN gibt es 4 weitere Standardfunktionen:

PEEK (a)	ist der Wert des durch a bezeichneten Speicherplatzes;
INP (a)	ist der am Port a anliegende 8-Bit-Wert;
CALL POKE (a1,a2)	ersetzt den Inhalt des Speicherplatzes a1 durch den Inhalt von a2;
CALL OUT (a1,a2)	gibt den Wert von a2 an den durch a1 festgelegten Port aus.

6.6 BLOCK DATA-Unterprogramme

BLOCK DATA-Unterprogramme dienen dazu, einfachen Variablen und Feldern, die in benannten COMMON-Bereichen liegen, Anfangswerte zuzuweisen. In Hauptprogrammen sowie FUNCTION- und SUB-

ROUTINE-Unterprogrammen ist dies nicht möglich. Variablen Feldern, die im unbenannten COMMON-Bereich liegen, können auf keine Art Anfangswerte zugewiesen werden.

Eine BLOCK DATA-Programmeinheit ist eine selbständige Programmeinheit. Ein auftretender COMMON-Bereich stimmt mit gleichnamigen COMMON-Bereichen in anderen Programmeinheiten überein. Zu gleichnamigen Größen anderer Art besteht keine Beziehung.

Allgemeine Form: BLOCK DATA [name]
Vereinbarungsanweisungen
END

Die Angabe eines Namens kann entfallen.
Es sind folgende Vereinbarungsanweisungen zugelassen:

COMMON, EQUIVALENCE, DIMENSION, DATA, Typanweisungen.

Alle auftretenden Variablen und Felder müssen in benannten COMMON-Bereichen liegen. Einer einfachen oder indizierten Variablen bzw. einem Feld dürfen erst Anfangswerte zugewiesen werden, wenn sie bzw. es als Element eines COMMON-Bereiches vereinbart ist (mit COMMON bzw. EQUIVALENCE). Wenn ein COMMON-Bereich-Name in einer BLOCK DATA-Programmeinheit auftritt, muß die Programmeinheit den gesamten COMMON-Bereich enthalten, d.h. auch die Elemente, denen keine Anfangswerte zugewiesen werden. Ein COMMON-Bereich kann nur durch eine BLOCK DATA-Programmeinheit initialisiert werden.

Beispiel
```
BLOCK DATA
LOGICAL A1
COMMON/BETA/B(3,3)/GAM/C(4),/XYZ/ZX(2,5)
COMMON/ALPHA/A1,C,E,D
DATA B/1.1,2.5,3.8,3*4.96,2*Ø.52,1.1/
1 C/1.2EØ,3*4.Ø/
DATA A1/.TRUE./,E/-5.6/
```

6.7 Einbinden von Unterprogrammen

Dieser Abschnitt beschreibt den normalen Unterprogramm-Aufruf, wie er vom FORTRAN-Compiler erzeugt wird. Damit wird die Zusammenarbeit von FORTRAN- und NichtFORTRAN-Programmen ermöglicht.

Ein UP-Aufruf ohne Parameter erzeugt einen einfachen CALL-Befehl. Das entsprechende UP wird über RET verlassen.

Ein UP-Aufruf mit Parameter ergibt eine komplexere Aufruffolge. Parameter werden immer durch ihre Adresse übergeben (Adresse des unteren Bytes des aktuellen Arguments). Es werden also unabhängig vom Typ immer zwei Bytes belegt.
Die Art der Übergabe hängt ab von der Anzahl der Parameter:
1. Bis zu drei Parameter werden über Registerpaare übergeben. Parameter 1 in HL, Parameter 2 in DE, Parameter 3 in BC.

2. Mehr als drei Parameter wie bei 1., nur daß BC die Anfangsadresse eines Datenblocks mit den restlichen Parametern ent-

hält. Dem UP muß außerdem noch die Anzahl übermittelt werden.

Wenn das UP mehr als drei Parameter erwartet und diese in ein lokales Datenfeld übertragen möchte, dann kann eine Systemroutine ¤AT benutzt werden. Vor dem Aufruf von ¤AT muß HL die Adresse des lokalen Datenblockes und A die Anzahl der zu übertragenden Daten enthalten. Das UP muß mit folgendem Vorspann versehen werden:

```
UP:  LD   (P1),HL
     LD   (P2),DE
     LD   HL,P3
     LD   A,ANZ-2
     CALL ¤AT
     :
     eigentliches Unterprogramm
     :
     RET

P1:  DS   2
P2:  DS   2
P3:  DS   2*(ANZ-2)
```

FORTRAN-Funktionen übergeben ihre Ergebnisse folgendermaßen: LOGICAL in A, INTEGER in HL, REAL in Speicher bei ¤AC, DOUBLE PRECISION in ¤DAC. ¤AC und ¤DAC sind die Adressen des Low-Bytes der Mantissen.

7. Aufbau von U880-FORTRAN-Programmen

Die Programmeinheiten müssen in U880-FORTRAN den folgenden Aufbau haben:

```
programmart   name

Typvereinbarungen
EXTERNAL-Anweisungen
DIMENSION-Anweisungen
COMMON-Anweisungen
EQUIVALENCE-Anweisungen
DATA-Anweisungen
Anweisungsfunktions-Definitionen

Ausführbare Anweisungen

END
```

Für programmart kann stehen

PROGRAM	für ein Hauptprogramm
SUBROUTINE	,dann folgt nach "name" die geklammerte Parameterliste
FUNCTION	,dann folgt nach "name" die geklammerte Parameterliste
BLOCK DATA	"name" kann fehlen.

Fehlt die PROGRAM-Anweisung, wird dem Hauptprogramm der Name ¤MAIN zugeordnet.

Der COMPLEX-Datentyp ist nicht implementiert.

8. Compilierung von FORTRAN-Programmen

8.1 Kommandoformat

Der Aufruf des FORTRAN-Compilers erfolgt durch:

```
% FORTRAN
```

Der Compiler meldet sich mit * , und ist bereit, ein FORTRAN-Quellprogramm zu compilieren. Das Quellprogramm ist mit EDIT erstellt und muß den in Abschnitt 2.2 angegebenen Bedingungen genügen. Der Quellfilename ist mit .S zu erweitern. Das allgemeine Kommandoformat ist:

```
* obj-dev/filename.ext,list-dev/filename.ext=quell-dev/filename.ext
  filename.ext
```

mit

obj-dev : Device (Gerät) für Ablage des Objektprogramms (Objektfile)

list-dev : Device für Ablage des Listingfiles

quell-dev: Device, auf dem sich das Quellprogramm (Quellfile) befindet.

Fehlen Devicenamen, werden die angegebenen Files überall gesucht.

filename.ext : Namen der Objekt-, Listings- und Quellfiles.

Die Filename-Erweiterungen (Extensions) können weggelassen werden. Es werden dann die Standarderweiterungen automatisch angefügt. S für Quellfile-, OBJ für Objektfile- und LST für Listingfileerweiterung.

Die Angaben links vom Gleichheitszeichen können teilweise oder ganz weggelassen werden. Es ergeben sich dann z.B. die folgenden Möglichkeiten:

* = FORT	Compiliert das Programm FORT.S und erzeugt den File FORT.OBJ ;
*,SYSLIST=FORT	Compiliert das Programm FORT.S und druckt das Listing. Es wird kein Objektfile erzeugt ;
*FORTOBJ=FORT	Compiliert FORT.S und legt das Objektprogramm unter FORTOBJ.OBJ ab ;
*FORT,FORT=FORT	Compiliert FORT.S und erzeugt FORT.OBJ und FORT.LST ;
*,=PORT	Compiliert FORT.S. Objekt- und Listingfile werden nicht erzeugt. Fehlermeldungen werden gemacht.

Es gibt eine Anzahl von 'Selektoren', mit denen die Compilierung beeinflußt werden kann. Jedem 'Selektor' wird ein - (Minuszeichen) vorangestellt.

Selektor	Wirkung
O	Druckt alle Listingadressen usw. in oktaler Form,
H	Druckt alle Listingadressen usw. in hexadezimaler Form (Standard),

N Der erzeugte Code wird nicht gelistet;

R Erzwingt Erzeugen eines Objektfiles;

L Erzwingt das Erzeugen eines Listingfiles;

P Jedes P erweitert den Stackbereich um 100(80)Bytes. Wird nur verwendet, wenn während der Compilierung Error 134 (Stack-Überlauf) auftritt.

Beispiele

*,SYSLIST=FORT-N	Compiliert FORT.S und gibt das Listing ohne Objektcode auf Drucker aus.
*=FORT-L	Compiliert FORT.S mit Erzeugung von FORT.OBJ und FORT.LST;
*=FORT-P-P	Compiliert FORT.S und erzeugt FORT.OBJ. Dem Compiler ist ein um 200 Bytes größerer Stackbereich zugeordnet.

Selektor Wirkung

M Der erzeugte Code ist in einer Form, wie er für das Laden in PROMs notwendig ist. Der erzeugte Code unterscheidet sich vom üblichen in:
1. FORMAT-Anweisungen liegen im Programmbereich und werden mit "JMP" übersprungen.
2. Parameterblöcke (für Unterprogrammaufrufe mit mehr als drei Parametern) werden zur Laufzeit initialisiert und nicht durch den Lader.

Bemerkung: Ist ein FORTRAN-Programm für EPROMs bestimmt, sollte beachtet werden:

1. DATA-Anweisungen können nicht benutzt werden, um RAM-Plätze zu initialisieren. Derartige Initialisierungen sind zur Laufzeit nicht vorhanden. Variable und Felder können während der Laufzeit durch Ergibt- oder READ-Anweisungen initialisiert werden.
2. FORMATe können während der Laufzeit nicht eingelesen werden (Abschnitt 5.4.7).
3. Wenn die I/O-Routinen der Standardbibliothek benutzt werden, können Diskfiles nur auf den logischen Einheiten 6, 7,8,9,10 eröffnet werden (Abschnitt 8.3). Werden andere logische Einheiten für Diskein- und -ausgaben benötigt, muß LUNTB mit den geeigneten Adressen für die Diskettentreiberroutinen versehen werden (Abschnitt 8.1).

Eine Bibliotheksroutine %INIT setzt den Stackpointer auf die Spitze des verfügbaren Speichers (wie durch das Betriebssystem (OS) angezeigt), bevor die Programmausführung beginnt. Der Aufruf lautet:

```
LD  BC, Rückkehradresse
JP  %INIT
```

Für eine andere Maschine muß diese Routine umgeschrieben werden!
Das Programm FORTRAN wird durch CTRL C verlassen. Es meldet sich das Betriebssystem mit "%".

8.2 Fehlermeldungen des FORTRAN-Compiler

Es werden zwei Fehlerarten unterschieden:

Warnungen: Bei dieser Fehlerart wird die Compilierung mit dem nächsten Objekt der Quellzeile fortgesetzt. Warnungsmeldungen wird das Prozentzeichen (%) vorangestellt.

Fatale Fehler: Tritt diese Fehlerart auf, ignoriert der Compiler den Rest der logischen Zeile, einschließlich der Fortsetzungszeilen. Fatalen Fehlern wird ein Fragezeichen (?) vorangestellt.

Bei der Fehlerausgabe wird die Zeilennummer mit angegeben.

Warnungen:

Ø Doppelte Anweisungsmarke
1 Regelwidriges DO-Ende
2 Blockname = Prozedurname
3 Mißbrauch von Feldnamen
4 COMMON-Namen-Behandlung
5 Falsche Zahl von Indizes
6 Feld mehrfach in einer EQUIVALENCE-Gruppe
7 Mehrfache EQUIVALENCE von COMMON
8 COMMON nach links erweitert
9 Nicht-COMMON-Variable in BLOCK DATA
10 Leere Liste für formatfreies WRITE
11 Nicht-ganzzahliger Ausdruck
12 Operandenmodus mit Operator nicht kompatibel
13 Operandenmischung nicht erlaubt
14 Fehlende ganzzahlige Variable
15 Fehlende Marke bei FORMAT
16 Wiederholungsfaktor Ø
17 Formatwert Ø
18 Zu tiefe Formatschachtelung
19 Marke keinem FORMAT zugeordnet
20 Ungültige Benutzung einer Marke
21 Anweisung nicht erreichbar
22 Fehlendes Ende eines DO
23 Codeausgabe in BLOCK DATA
24 Nichtdefinierte Marke
25 RETURN in einem Hauptprogramm
26 Statusfehler bei READ
27 Falsche Operandenverwendung
28 Funktion ohne Parameter
29 Überlauf bei hexadezimaler Konstante
30 Division durch Ø
32 Feldname wird erwartet
33 Falsches Argument bei ENCODE/DECODE

Fatale Fehler:

100 Falsche Marke
101 Anweisung nicht deutbar
102 Falscher Anweisungsabschluß
103 Falsche DO-Schachtelung
104 Falsche Datenkonstante
105 Fehlender Name

106 Falscher Prozedurname
107 Falsche DATA-Konstante oder -Wiederholfaktor
108 Falsche Anzahl von DATA-Konstanten
109 Falsche ganzzahlige Konstante
110 Ungültige Marke
111 Kein Variablenname
112 Falscher Operator in logischem Ausdruck
113 Überlauf des Datenpuffers
114 Zeichenkette zu lang
115 Falsches Element in I/O-Liste
116 Unrichtige DO-Schachtelung
117 Name zu lang
118 Falscher Operator
119 Nichtpassende Klammerung
120 Aufeinanderfolgende Operatoren
121 Falsche Indexverwendung
122 Nichtzugelassener ganzzahliger Wert
123 Falsche Zeichenkette
124 DO-Bezug rückwärts
125 Falscher Anweisungsfunktionsname
126 Falsches Zeichen
127 Anweisung außerhalb der Programmfolge
128 Fehlender ganzzahliger Wert
129 Falscher logischer Operator
130 Falsche Angabe nach INTEGER, REAL oder LOGICAL
131 Vorzeitiges EOF auf Eingabegerät
132 Nichtzugelassene gemischte Operation
133 Funktionsaufruf ohne Parameter
134 Stack-Überlauf
135 Falsche Anweisung nach logischem IF

9. Binden von FORTRAN-Objektprogrammen

9.1 Der Binder (FORTRAN-LINKER) FLINK

Der Aufruf des Binders (Linkers) erfolgt aus dem Betriebssystem durch %FLINK gefolgt von, jeweils durch Komma getrennten, Filenamen der zu bindenden Objektfiles. Wie beim Compiler können zusätzlich zum Filenamen das entsprechende Eingabegerät (Input devices) sowie Selektoren angegeben werden.

```
obj-dev1/filename.ext-sel1,obj-dev2/filename.ext-sel2,....
```

Fehlt die Angabe des Eingabe-Geräts (Input Device) für ein File, wird es gesucht. Fehlt die Erweiterung, wird automatisch .OBJ angenommen. Nach jeder geschriebenen Zeile lädt oder sucht (siehe S weiter unten) FLINK die angegebenen Files. Nach Beendigung dieses Prozesses werden alle undefinierten Symbole gefolgt von einem Stern aufgelistet. z.B.

```
%FLINK
*MAIN
 SUBR1*     SUBR1 ist nicht definiert
*SUBR1
*-G         (Beginn der Ausführung - siehe unten)
```

Um ein FORTRAN-Programm und Unterprogramme auszuführen, wird an die Liste der Filenamen ein -G angehängt (Beginn der Ausführung). Benötigt das Programm eine FORTRAN-Bibliotheks-Routine, wird so automatisch in FORLIB.OBJ gesucht, bevor die Ausführung beginnt. Sollen zuerst nutzereigene Bibliotheken durchsucht werden, sollten die mit -S verlängerten Filenamen an das Ende der Kommandozeile angefügt werden.

Selektor	Wirkung
R	Setzt FLINK in den Anfangszustand. -R wirkt sofort, wenn es in der Kommandozeile auftritt.
E oder E:name	Bewirkt, daß FLINK verlassen wird. FORLIB.OBJ wird durchsucht, um noch existierende undefinierte globale Größen zu befriedigen. Die Form E:name (wobei name ein vorher in einem Modul definiertes globales Symbol ist) benutzt name als Startadresse des Programms. Mit -E wird ein Programm geladen. Es kann dann mit IMAGE als ladbares Programm zur Diskette gespeichert werden.
G oder G:name	Startet die Ausführung des Programms, sobald die momentane Kommandozeile interpretiert worden ist. FORLIB.OBJ wird durchsucht, um eventuell existierende undefinierte globale Symbole zu definieren. Vor der Ausführung druckt FLINK drei Zahlen nebeneinander. [Startadresse Adresse des nächsten verfügbaren Bytes Seitenanzahl] BEGIN EXECUTION Die Form G:name entspricht E:name
U	Listet alle undefinierten globalen Symbole;
M	Listet alle definierten globalen Symbole und ihre Werte, sowie die undefinierten gefolgt von einem *.
S	Durchsucht den Filenamen unmittelbar vor -S um undefinierte globale Symbole zu definieren.

-U und -M drucken auch den Anfang und das Ende von Programm- und Datenbereich.

```
DATA      100     200
PROGRAM   1000    2000
```

Die Programminformation wird nur angegeben, wenn ein -D angegeben wurde, sonst wird das Programm im Datenbereich gespeichert.

N	Wenn filename-N angegeben wird, wird das Programm auf Diskette gerettet unter dem gewählten Namen. Es muß in der Kommandozeile ein -E oder -G gegeben worden sein. Wenn nötig, wird ein Sprung zum Start des Programms eingefügt.

P und D -P und -D setzen die Anfänge für das nächste zu ladende Programm. Die Wirkung von -P und -D tritt ein, wenn sie in der Kommandozeile auftreten. Sie haben keinen Einfluß auf schon geladene Programme. Die Form ist -P:Adresse oder -D:Adresse, wobei von FLINK drei Plätze für einen Sprung zur Startadresse freigelassen werden.

Ist -D nicht angegeben, werden die Programme wie üblich geladen, mit der Ausnahme, daß eines die Möglichkeit hat, die Basisadresse festzulegen. Ist ein -D gegeben, werden alle DATA und COMMON-Bereiche geladen, beginnend am Datenursprung. Der Programmbereich beginnt am Programmursprung.

Beispiel:
```
*-P:2ØØ,FOO
DATA    2ØØ    3ØØ
*-P
*-P:2ØØ -D:4ØØ,FOO
DATA    4ØØ    48Ø
PROG    2ØØ    28Ø
```

FLINK-Beispiel

```
*FLINK
*BEISP1, BEISP2-G
[3Ø4F    3ØAC  49]
[BEGIN   EXECUTION]
```

9.2 FLINK-Fehlermeldungen

?NO START ADDRESS	Ein -G ist angegeben, aber kein Hauptprogramm vorhanden
?LOADING ERROR	Das letzte Eingabefile war kein exaktes FLINK-Objektfile
?FATAL TABLE COLLISION	Verfügbarer Speicher zu klein
?COMMAND ERROR	Nicht deutbares FLINK-Kommando
?FILE NOT FOUND	Ein File aus der Kommandozeile existiert nicht
%2ND COMMON LARGER/XXXXXX/	Die erste Definition des COMMON-Blocks war nicht die größte.
?MULT.DEF.GLOBAL YYYYYY	Mehr als eine Definition für ein globales Symbol
?OUT OF MEMORY	Verfügbarer Speicher zu klein
?<file> NOT FOUND	Wie ?FILE NOT FOUND, Filename wird gedruckt
%OVERLAYING PROGRAM / DATA AREA	Ein -D oder -P zerstört schon gelegene Daten

Bemerkung: Wenn ein -D gegeben ist und der Datenanfang vor dem Programmbereich liegt, muß es sicher sein, daß ausreichend Platz für Daten vorhanden ist. Dies trifft besonders auf den Disktreiber für FORTRAN zu, der %MEMRY benutzt, um die Diskpuffer und FCBs Speicherbereichen zuzuordnen.

?INTERSECTING PROGRAM / DATA AREA

Programm- und Datenbereich überschneiden sich und eine Adresse oder Eintrittspunkt liegt in der Überschneidung

?START SYMBOL-<name> -UNDEFINED

Nach -E: oder -G: ist das angegebene Symbol nicht definiert

?ORIGIN ABOVE / BELOW LOADER MEMORY, MOVE ANYWAY(Y OR N)?

Nach einem -E oder -G hat entweder der Daten- oder der Programmbereich einen Anfang oder ein Ende außerhalb des Ladespeichers. Bei Y verschiebt FLINK den Bereich und macht weiter. Bei jedem anderen Zeichen wird FLINK verlassen. Ist -N gegeben, wird in beiden Fällen das Speicherbild gerettet.

?CAN'T SAVE OBJECT FILE

Diskettenfehler

Ist das Symbol %MEMRY durch ein geladenes Programm definiert, dann speichert FLINK die Adresse des ersten freien Platzes in %MEMRY.

9.3 Laufzeit-Fehlermeldungen

Warnungen:

IB	Eingabepuffergrenze überschritten
TL	In FORMAT zu viele linke Klammern
OB	Ausgabepuffergrenze überschritten
DE	Überlauf des Dezimalexponenten (>99)
IS	Integergröße zu groß
BE	Überlauf des Binärexponenten
IN	Eingabe-Record zu lang
CN	Überlauf bei Konvertierung REAL → INTEGER
SN	Argument von Sinus zu groß
A2	Beide Argumente von ATAN2 sind Ø
IO	Unzulässige I/O-Operation
BI	Puffergröße während binärer I/O-Operation überschritten
RC	Negativer Wiederholfaktor in FORMAT

Fatale Fehler:

FD	Unzulässiges Format in FORMAT
FO	Formatbreite ist Null
MP	Fehlender Punkt in FORMAT
FW	Formatbreite zu klein
IT	I/O-Übertragungsfehler

ML	Fehlende linke Klammern in FORMAT
DZ	Division durch Null bei REAL oder INTEGER
LG	Unzulässiges Argument bei LOG (Negative oder Null)
SQ	Unzulässiges Argument bei SQRT
DT	Datentyp stimmt nicht mit FORMAT-Festlegung überein
EF	EOF durch READ erreicht

Die Angabe der Fehler erfolgt in der Form: ##IB##

Fatale Fehler führen ins Betriebssystem zurück. Nach 20 Warnungen wird ebenfalls ins Betriebssystem zurückgegangen.

10. Standardfunktionen

Name	Definition	AP	TP	TF	Bemerkungen
ABS	f(p)=/p/	1	R	R	Absolutbetrag
IABS		1	I	I	
DABS		1	D	D	
INT	f(p)=[p]	1	R	I	[p]=sgn(p)*(größte ganze Zahl≤/p/)
AINT		1	R	R	
IDINT		1	D	D	
MOD	f(p1,p2)=	2	I	I	p2 ≠ Ø, Divisionsrest
AMOD	p1-[p1/p2]*p2	2	R	R	p2 ≠ Ø
MAXØ	f(p1,p2,...,pn)= max(p1,p2,...,pn)	≥2	I	I	
MAX1			R	I	
AMAXØ			I	R	
AMAX1			R	R	
DMAX1			D	D	
MINØ	f(p1,p2,...,pn)= min(p1,p2,...,pn)	≥2	I	I	
MIN1			R	I	
AMINØ			I	R	
AMIN1			R	R	
DMIN1			D	D	
FLOAT	f(I)=R	1	I	R	Konvertierung: ganzzahlig → reell
IFIX	f(p)=[p]		R	I	Konvertierung: reell → ganzzahlig
SIGN	f(p1,p2)=sign(p2)*/p1/	2	R	R	Vorzeichenübertragung
DIM	f(p1,p2)=	2	R	R	Schwache Differenz
IDIM	p1-min(p1,p2)	2	I	I	
SNGL	f(D)=R	1	D	R	Konvertierung: doppeltr.→reell
DBLE	f(R)=D	1	R	D	Konvertierung: reell → doppeltr.

In-line-Standardfunktionen
Abkürzungen: AP,TP:Anzahl bzw. Typ der Parameter, TF:Typ der Funktion, I ganzzahlig, R reell, D doppeltgenau

Name	Definition	AP	TP	TF	Bemerkungen
EXP	f(p) =	1	R	R	
DEXP	1**p	1	D	D	
ALOG	f(p) =	1	R	R	p > Ø
DLOG	ln(p)	1	D	D	p > Ø
ALOG1Ø	f(p) =	1	R	R	p > Ø
DLOG1Ø	lg(p)	1	D	D	p > Ø
SIN	f(p) =	1	R	R	
DSIN	sin(p)	1	D	D	
COS	f(p) =	1	R	R	
DCOS	cos(p)	1	D	D	
TANH	f(p)=tanh(p)	1	R	R	Tangens hyperbolicus
ATAN	f(p) =	1	R	R	Arcustangens
DATAN	arctan(p)	1	D	D	
ATAN2	f(p1,p2)=	2	R	R	$-\widetilde{\pi} < f(p1,p2) \leq +\widetilde{\pi}$
DATAN2	arctan(p1/p2)	2	D	D	
SQRT	f(p) =	1	R	R	p ≥ Ø
DSQRT	$\sqrt{p}$	1	D	D	p ≥ Ø
DMOD	f(p1,p2)=p1-[p1/p2]*p2	2	D	D	p2≠Ø, Divisionsrest

Out-of-line-Standardfunktionen

III-12-12 Kv 120/86